68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30:-SEK

$2.95 USA

OS-9 Atari Amiga Mac S-50

6800 6809 68000 68008 68010 68020 68030

The Magazine for Motorola CPU Devices *For Over a Decade!*

This Issue:

OS-9 SK*DOS Atari Amiga FLEX Macintosh

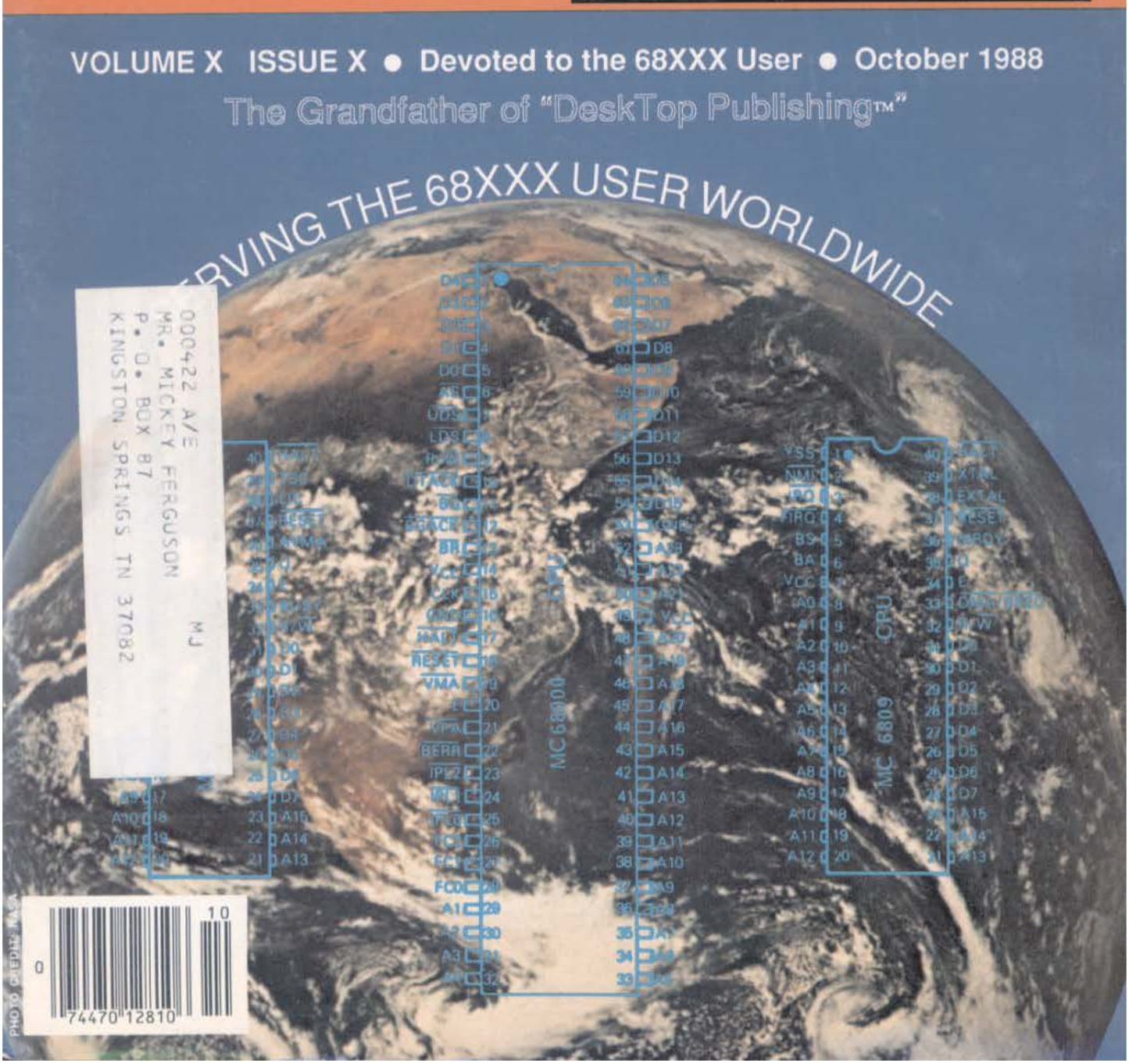
A User Contributor Journal

And Lots More!

VOLUME X ISSUE X • Devoted to the 68XXX User • October 1988

The Grandfather of "DeskTop Publishing™"

SERVING THE 68XXX USER WORLDWIDE



0 74470 12810 10

PHOTO CREDIT: NASA

WHO DO YOU CALL WHEN YOUR DEBUGGER WON'T DEBUG?

The problem with most real-time operating systems is simple, they're not an integrated solution. You end up dealing with a multitude of suppliers for languages, compilers, debuggers and other important development tools. And when something does go wrong, it can be a frustrating experience trying to straighten out the mess.

Why Not Try the Microware One-Stop Total Solution?

Microware's OS-9 Real-Time Operating System is a total integrated software system, not just a kernel. We offer an extensive set of development tools, languages, I/O and Kernel options. ***And this total integrated solution is entirely designed, built and supported by the same expert Microware team.***

Modularity Lets YOU Choose Just What You Need.

The modular design of OS-9 allows our Operating System to adapt as your requirements change. OS-9 can support a complete spectrum of applications — from embedded ROM-based code in board-level products all the way up to large-scale systems.

The OS-9 Success Kit

A Total Integrated Solution for Your Next Project

Development Tools:
- C Source Level Debugger
- Symbolic Debugger
- System State Debugger
- uMACS Text Editor
- Electronic Mail
- Communications
- Super Shell

Kernel Options:
- MMU (Security Protection) Support
- Math Coprocessor Support

Languages:
- C*
- Basic
- Pascal
- Fortran
- Ada**
- Assembler*

I/O Options:
- SCSI, SASI & SMD Disks
- 3-, 5-, 8-inch Diskettes
- Magnetic Tape
- Ethernet - TCP/IP
- Arcnet - OS-9/Net

*Resident or UNIX versions available
**VAX hosted

Support is Part of the Package.

Microware is proudly setting the industry's standard for customer support. You'll find professional and comprehensive technical documentation and a Customer Hotline staffed by courteous and authoritative software engineers.

So stop messing with simple kernels and independent suppliers. Call Microware today and find out more about the "One-Stop Integrated Solution" with OS-9!

microware® OS-9

Microware Systems Corporation
1900 N.W. 114th Street
Des Moines, Iowa 50322
Phone: 515/224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, California 95054
Phone: 408/980-0201

Microware Japan Ltd.
41-19 Honcho 4-Chome
Funabashi City
Chiba 273, Japan
Phone: 0474 (22) 1747

THE SOFT CENTRE OPEN U.S. OFFICES

As a result of increased sales in the U.K. and Europe of some of the most exciting products to hit the OS-9 world, the Soft Centre announces the opening of Windsor Systems. Windsor Systems is owned by Steven Weller, Steve is a senior software engineer and has been employed by the Soft Centre for the last four years.

Software products now available in North America with full support include:-

GKS ... Graphical Kernel System to level 2c.

DISK CACHING ... make your computer run three times faster, this is the typical performance improvement on a 68020 Computer with hard disk.

SCSI ... Device Driver System for using SCSI with OS-9, offering separate logical and physical drivers,the SCSI driver system creates a flexible interface for all your SCSI drivers. Supports SCSI commands disconnect and reconnect.

TAPE ARCHIVING ... Integrated bulk storage archiving utility, incorporating UNIX Tar and Cpio compatibility.

VBF ... Variable Block File Manager, a communications orientated file manager which brings out the best in multichannel intelligent interface cards.

MCF ... Multi-Character File Manager is the same as VBF, but additionally allows the use of line editing functions on read line and write line.

VIVANET ... Vivanet is a serial port driver for the Network File Manager.

MATH ... Math trap handler and math library for the 68008, 68000 and 68010 using the MC68881 as a peripheral.

For more information contact:-

Windsor Systems
2407 Lime Kiln Lane
Louisville
Kentucky 40222
U.S.A.

Tel: 502-425-9560
Fax: 502-425-6853

The Soft Centre
Software House
Burr Street
LUTON Bedfordshire
LU2 0HN England

Tel: 011-44-582-405511
Fax: 011-44-582-456521

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Cheryl Hodge

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law
Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

---

68 Micro Journal (ISSN 0194-5025) is published 12 times a year by Computer Publishing Inc. Second Class Postage paid at Hixson, TN. and additional entries. POSTMASTER: send address changes to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - donot format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9
68XXX
Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED **SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO
PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

MUSTANG -020 Super SBC™

DATA-COMP Proudly Presents the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel. Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020 systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisitioncost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost of less than $65 per port, in multiples of 8 port expansion options.

The SBC fullypopulated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5- 25 Mhz systems, call for special prices

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division

CPI A Decade of Quality Service™

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.3*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
            register long i;
            for (i=0; i < 9999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 — 4.5 MIPS,
Burst to 8 - 10 MIPS: Motorola Specs

OS-9

OS-9 Professional Ver *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Onyx/soft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adaptor Cards below RS232 Adapter	165.00
Each card supports 4 additional ser. ports (total of 36 serial ports supported)	
60 line Parallel I/O card	398.00
Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	
Prototype Board	75.00
area for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	
SBC-AN	475.00
Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call.	
LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00
Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Xtra 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

The PRO!

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note: all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

Save $1000.00
Complete
25 Mbyte HD System
$4299.80
85Mbyte HD System
$5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division
A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

C *The C Programmers Reference Source. Always Right On Target!*

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter continues the discussion of dbug, a C debugging package. It is a useful tool for debugging and testing C programs. It was developed by Fred Fish, who placed it into the public domain. The C code for the dbug package appears in the previous chapters. This chapter contains the code for analyze.c, an extension to the dbug package which provides a form of program profiling, provided by Binayak Banerjee, who also placed it into the public domain.

ANALYZE.C

The dbug routines create a file called dbugmon.out (by default). This is an ASCII file containing lines of the following forms:

<function-name> E <time-entered>
<function-name> X <time-exited>

Analyze reads this file, and produces a report on standard output.

Profiling is enabled through the 'g' flag. It can take a list of procedure names for which profiling is enabled. By default, it profiles all procedures.

The only system-dependent element is the routine Clock() at the end of dbug.c. This function must return the user time in milliseconds.

An example of the report generated follows:

Profile of Execution
Execution times are in milliseconds

	Calls		Time		
Function	Times Called	Percentage of total	Time Spent in Function	Percentage of total	Importance
factorial	5	83.33	30	100.00	8333
main	1	16.67	0	0.00	0
Totals	6	100.00	30	100.00	

The Importance column is a metric obtained by multiplying the percentage of the calls and the percentage of the time. Functions with higher 'importance' benefit the most from being sped up.

```
/**
 * Analyze profile file written by dbug routines with
profiling enabled.
 *
 * Binayak Banerjee
 */

#include <stdio.h>

#ifdef BSD
#include <sysexits.h>
#else
#define EX_SOFTWARE 1
#define EX_DATAERR 1
#define EX_USAGE 1
#define EX_OSERR 1
#define EX_IOERR 1
#define EX_OK 0
#endif

#ifndef DBUG_OFF
#include "dbug.h"
#else
#define DBUG_ENTER(a1)
#define DBUG_RETURN(a1) return(a1)
#define DBUG_VOID_RETURN return
#define DBUG_EXECUTE(keyword,a1)
#define DBUG_PUSH(a1)
#define DBUG_POP()
#define DBUG_PROCESS(a1)
#define DBUG_PRINT(x,y)
#define DBUG_FILE (stderr)
#endif
```

```
    #define __MERF_OO_ "%s: Malloc Failed in %s: %d\n"

    #define Nil(Typ)        ((Typ *) 0)     /* Make Lint
happy */

    #define MALLOC(Ptr,Num,Typ) do  /* Malloc w/error
checking & exit */ \
            if ((Ptr = (Typ *)malloc((Num)*(sizeof(Typ))))
== Nil(Typ)) \
            {fprintf(stderr,__MERF_OO_,my_name,__FILE__,__LINE__);\
                    exit(EX_OSERR);} while(0)

    #define Malloc(Ptr,Num,Typ) do  /* Weaker version of
above */\
            if ((Ptr = (Typ *)malloc((Num)*(sizeof(Typ))))
== Nil(Typ)) \
            fprintf(stderr,__MERF_OO_,my_name,__FILE__,__LINE__);\
                    while(0)

    #define FILEOPEN(Fp,Fn,Mod) do  /* File open with
error exit */ \
            if((Fp = fopen(Fn,Mod)) == Nil(FILE))\
                    {fprintf(stderr,"%s: Couldn't open
%s\n",my_name,Fn);\
                    exit(EX_IOERR);} while(0)

    #define Fileopen(Fp,Fn,Mod) do  /* Weaker version of
above */ \
            if((Fp = fopen(Fn,Mod)) == Nil(FILE)) \
                    fprintf(stderr,"%s: Couldn't open
%s\n",my_name,Fn);\
            while(0)

    extern char *my_name;   /* The name that this was
called as */

    char *my_name;
    int verbose;
    unsigned int stacktop = 0;      /* Lowest stack
position is a dummy */
    unsigned long tot_calls = 0;
    unsigned long tot_time = 0;

    /*
     * Structure of the stack.
     */

    #define MAXPROCS        1000    /* Maximum number of
function calls */
    #define PRO_FILE        "dbugmon.out" /* Default output
file name */
    #define STACKSIZ        100     /* Maximum function
nesting */

    struct stack_t
    {
        unsigned int pos;               /* which function?
*/
        unsigned long time;             /* Time that this
was entered */
        unsigned long children;         /* Time spent in
called funcs */
    }
        fn_stack[STACKSIZ+1];

    /*
     * Push - Push the given record on the stack.
     */

    void
    push(name_pos, time_entered)
    register unsigned int name_pos;
    register unsigned long time_entered;
    {
        register struct stack_t *t;

        DBUG_ENTER("push");
        if (++stacktdp > STACKSIZ)
        {
            fprintf(DBUG_FILE, "%s: stack overflow
(%s:%d)\n",
                    my_name, __FILE__, __LINE__);
            exit(EX_SOFTWARE);
        }
        DBUG_PRINT("push", ("%d %ld", name_pos,
time_entered));
        t = &fn_stack[stacktop];
        t->pos = name_pos;
        t->time = time_entered;
        t->children = 0;
        DBUG_VOID_RETURN;
    }


    /*
     * Pop - pop the top item off the stack, assigning the
field values
     * to the arguments. Returns 0 on stack underflow, or on
popping first
     * item off stack.
     */

    unsigned int
    pop(name_pos, time_entered, child_time)
    register unsigned int *name_pos;
    register unsigned long *child_time;
    register unsigned long *time_entered;
    {
        register struct stack_t *temp;

        DBUG_ENTER("pop");
        if (stacktop < 1)
            DBUG_RETURN(0);
        temp = &fn_stack[stacktop];
        *name_pos = temp->pos;
        *time_entered = temp->time;
        *child_time = temp->children;
        DBUG_PRINT("pop", ("%d %d %d", *name_pos,
*time_entered, *child_time));
        DBUG_RETURN(stacktop--);
    }


    /*
     * We keep the function info in another array (a simple
symbol table)
     */

    struct module_t
    {
        char *name;
        unsigned long m_calls;
        unsigned long m_time;
    }
        modules[MAXPROCS];

    /*
     * We keep a binary search tree in order to look up
function names
     * quickly and sort them at the end.
     */

    struct
    {
        unsigned int lchild;    /* Index of left subtree */
        unsigned int pos;       /* Index of module_name
entry */
        unsigned int rchild;    /* Index of right subtree */
    }
        s_table[MAXPROCS];

    unsigned int n_items = 0;       /* No. of items in the
array so far */

    /*
     * Need a function to allocate space for a string and
squirrel it away.
     */

    char *
    strsave(s)
```

```
    char *s;
    {
        extern char *malloc ();
        register char *retval;
        regieter unsigned int len;

        DBUG_ENTER("strsave");
        DBUG_PRINT("strsave", ("%s", s));
        if (s == Nil(char) || (!(len = strlen(s))))
            DBUG_RETURN(Nil(char));
        MALLOC(retval, ++len, char);
        strcpy(retval, s);
        DBUG_RETURN(retval);
    }


    /*
     * add() - adds m_name to the table (if not already
there), and returns
     * the index of its location in the table.  Checks
s_table (which is a
     * binary search tree) to see whether or not it should
be added.
     */

    unsigned int
    add(m_name)
    char *m_name;
    {
        register int cmp;
        register unsigned int ind = 0;

        DBUG_ENTER("add");
        if (!n_items) /* First item to be added */
        {
            s_table[0].pos = ind;
            s_table[0].lchild = s_table[0].rchild =
MAXPROCS;
    addit:
            modules[n_items].name = strsave(m_name);
            modules[n_items].m_time =
modules[n_items].m_calls = 0;
            DBUG_RETURN(n_items++);
        }
        while (cmp = strcmp(m_name, modules[ind].name))
        {
            if (cmp < 0)
            {  /* In left subtree */
                if (s_table[ind].lchild == MAXPROCS)
                {
                    /* Add as left child */
                    if (n_items >= MAXPROCS)
                    {
                        fprintf(DBUG_FILE,
                            "%s: Too many functions being
profiled\n", my_name);
                        exit(EX_SOFTWARE);
                    }
                    s_table[n_items].pos =
s_table[ind].lchild = n_items;
                    s_table[n_items].lchild =
s_table[n_items].rchild = MAXPROCS;
    #ifdef notdef
                    modules[n_items].name =
strsave(m_name);
                    modules[n_items].m_time =
modules[n_items].m_calls = 0;
                    DBUG_RETURN(n_items++);
    #else
                    goto addit;
    #endif
                }
                ind = s_table[ind].lchild; /* else
traverse l-tree */
            }
            else
            {
                if (s_table[ind].rchild == MAXPROCS)
                {
                    /* Add as right child */
                    if (n_items >= MAXPROCS)
                    {
                        fprintf(DBUG_FILE,
                            "%s: Too many functions being
profiled\n", my_name);
                        exit(EX_SOFTWARE);
                    }
                    s_table[n_items].pos =
s_table[ind].rchild = n_items;
                    s_table[n_items].lchild =
s_table[n_items].rchild = MAXPROCS;
    #ifdef notdef
                    modules[n_items].name = strsave(m_name);
                    modules[n_items].m_time =
modules[n_items].m_calls = 0;
                    DBUG_RETURN(n_items++);
    #else
                    goto addit;
    #endif
                }
                ind = s_table[ind].rchild; /* else traverse
r-tree */
            }
        }
        DBUG_RETURN(ind);
    }

    /*
     * process() - process the input file, filling in the
modules table.
     */

    void
    process(inf)
    FILE *inf;
    {
        char buf[BUFSIZ];
        char fn_name[25];       /* Max length of fn_name */
        char fn_what[2];
        struct stack_t *t;
        unsigned int oldpos;
        unsigned int pos;
        unsigned long fn_time;
        unsigned long oldchild;
        unsigned long oldtime;
        unsigned long time;

        DBUG_ENTER("process");
        while (fgets(buf, BUFSIZ, inf))
        {
            sscanf(buf, "%24s %1s %ld", fn_name, fn_what,
&fn_time);
            pos = add(fn_name);
            DBUG_PRINT("enter", ("%s %s %d", fn_name,
fn_what, fn_time));
            if (fn_what[0] == 'E')
                push(pos, fn_time);
            else
            {
                /*
                 * An exited function implies that all
stacked
                 * functions are also exited, until the
matching
                 * function is found on the stack.
                 */
                while ( pop(&oldpos, &oldtime, &oldchild) )
                {
                    DBUG_PRINT("popped", ("%d %d", oldtime,
oldchild));
                    time = fn_time - oldtime;
                    t = &fn_stack[stacktop];
                    t->children += time;
                    DBUG_PRINT("update", ("%s", modules[t-
>pos].name));
                    DBUG_PRINT("update", ("%d", t->chil-
dren));
                    time -= oldchild;
                    modules[oldpos].m_time += time;
                    modules[oldpos].m_calls++;
                    tot_time += time;
                    tot_calls++;
```

```
                    if (pos == oldpos)
                        goto next_line; /* Should be a
break2 */
                }
                /*
                 * Assume that item seen started at time
0.
                 * (True for function main).  But initial-
ize
                 * it so that it works the next time too.
                 */
                t = &fn_stack[stacktop];
                time = fn_time - t->time - t->children;
                t->time = fn_time;
                t->children = 0;
                modules[pos].m_time += time;
                modules[pos].m_calls++;
                tot_time += time;
                tot_calls++;
            }
    next_line:;
        }
        /*
         * Now, we've hit eof.  If we still have stuff
stacked, then we
         * assume that the user called exit, so give
everything the exited
         * time of fn_time.
         */
        while (pop(&oldpos, &oldtime, &oldchild))
        {
            time = fn_time - oldtime;
            t = &fn_stack[stacktop];
            t->children += time;
            time += oldchild;
            modules[oldpos].m_time += time;
            modules[oldpos].m_calls++;
            tot_time += time;
            tot_calls++;
        }
        DBUG_VOID_RETURN;
    }


    /*
     * out_header() - print out the header of the report.
     */

    void
    out_header(outf)
    FILE *outf;
    {
        DBUG_ENTER("out_header");
        if (verbose)
        {
            fprintf(outf, "Profile of Execution\n");
            fprintf(outf, "Execution times are in
milliseconds\n\n");
            fprintf(outf, "    Calls\t\t\t    Time\n");
            fprintf(outf, "    -\t\t\t    -\n");
            fprintf(outf, "Times\tPercentage\tTime
Spent\tPercentage\n");
            fprintf(outf,
               "Called\tof total\tin Function\tof total
Importance\tFunction\n");
            fprintf(outf,
           "------\t----------\t----------\t----------
----------=\t--------\t\n");
        }
        else
            fprintf(outf,
                "   %%time     sec    #call ms/call
%%calls  weight   name\n");
        DBUG_VOID_RETURN;
    }


    /*
     * out_trailer() - writes out the summary line of the
report.
     */

    void
    out_trailer(outf, sum_calls, sum_time)
    FILE *outf;
    unsigned long sum_calls;
    unsigned long sum_time;
    {
        DBUG_ENTER("out_trailer");
        if (verbose)
        {
            fprintf(outf,
            "-----\t--------\t---------\t--------\n");
            fprintf(outf, "%6d\t%10.2f\t%11d\t%10.2f\t\t%-
15s\n",
                sum_calls, 100.0, sum_time, 100.0, "To-
tals");
        }
        DBUG_VOID_RETURN;
    }


    /*
     * out_item() - prints out the output line for a single
entry,
     * and sets the calls and time fields appropriately.
     */

    void
    out_item(outf, m, called, timed)
    FILE *outf;
    register struct module_t *m;
    unsigned long *called;
    unsigned long *timed;
    {
        char *name = m->name;
        double ftime;
        double ms_per_call;
        double per_calls = 0.0;
        double per_time = 0.0;
        register unsigned int calls = m->m_calls;
        register unsigned long time = m->m_time;
        unsigned int import;

        DBUG_ENTER("out_item");
        if (tot_time > 0)
            per_time = (double) (time * 100) / (double)
tot_time;
        if (tot_calls > 0)
            per_calls = (double) (calls * 100) / (double)
tot_calls;
        import = (unsigned int) (per_time * per_calls);
        if (verbose)
        {
            fprintf(outf, "%6d\t%10.2f\t%11d\t%10.2f
%10d\t%-15s\n",
                calls, per_calls, time, per_time, import,
name);
        }
        else
        {
            ms_per_call = time;
            ms_per_call /= calls;
            ftime = time;
            ftime /= 1000;
            fprintf(outf, "%8.2f%8.3f%8u%8.3f%8.2f%8u  %-
s\n",
                per_time, ftime, calls, ms_per_call,
per_calls, import, name);
        }
        *called = calls;
        *timed = time;
        DBUG_VOID_RETURN;
    }


    /*
     * out_body(outf,root,s_calls,s_time) - Performs an
inorder traversal
     * on the binary search tree (root).  Calls out_item to
actually print
     * the item out.
     */
```

```
void
out_body(outf, root, s_calls, s_time)
FILE *outf;
register unsigned int root;
register unsigned long *s_calls;
register unsigned long *s_time;
{
    unsigned long calls;
    unsigned long time;

    DBUG_ENTER("out_body");
    DBUG_PRINT("out_body", ("%d,%d", *s_calls,
*s_time));
    if (root == MAXPROCS)
    {
        DBUG_PRINT("out_body", ("%d,%d", *s_calls,
*s_time));
        DBUG_VOID_RETURN;
    }
    while (root != MAXPROCS)
    {
        out_body(outf, s_table[root].lchild, s_calls,
s_time);
        out_item(outf, &modules[s_table[root].pos],
&calls, &time);
        DBUG_PRINT("out_body", ("- %d - %d -", calls,
time));
        *s_calls += calls;
        *s_time += time;
        root = s_table[root].rchild;
    }
    DBUG_PRINT("out_body", ("%d,%d", *s_calls,
*s_time));
    DBUG_VOID_RETURN;
}


/*
 * output() - print out sorted output report on outf.
 */

void
output(outf)
FILE *outf;
{
    unsigned long sum_calls;
    unsigned long sum_time;

    sum_calls = sum_time = 0;
    DBUG_ENTER("output");
    if (!n_items)
    {
        fprintf(outf, "%s: No functions to trace\n",
my_name);
        exit(EX_DATAERR);
    }
    out_header(outf);
    out_body(outf, 0, &sum_calls, &sum_time);
    out_trailer(outf, sum_calls, sum_time);
    DBUG_VOID_RETURN;
}


#define usage() fprintf(DBUG_FILE,"Usage: %s [-v]
[prof-file]\n",my_name)

main(argc, argv, environ)
int argc;
char *argv[], *environ[];
{
    FILE *infile;
    FILE *outfile = stdout;
    extern char *optarg;
    extern int getopt();
    extern int optind;
    int badflg = 0;
    register int c;

    DBUG_ENTER("main");
    DBUG_PROCESS(argv[0]);
    my_name = argv[0];
    while ((c = getopt(argc, argv, "#:v")) != EOF)
    {
        switch (c)
        {
        case '#': /* Debugging Macro enable */
            DBUG_PUSH(optarg);
            break;
        case 'v': /* Verbose mode */
            verbose++;
            break;
        default:
            badflg++;
            break;
        }
    }
    if (badflg)
    {
        usage();
        DBUG_RETURN(EX_USAGE);
    }
    if (optind < argc)
        FILEOPEN(infile, argv[optind], "r");
    else
        FILEOPEN(infile, PRO_FILE, "r");
    process(infile);
    output(outfile);
    DBUG_RETURN(EX_OK);
}
```

EXAMPLE C PROGRAM

Following is this month's example C program: it plays the game of pong under os-9, requiring only cursor setting capabilities of a terminal.

```
/*
 *    Rewrite of R. M. Halstead's Pong in C
 *    with conversions for Televideo terminal.
 *
 *    This version does not require the
 *    terminal to have a special graphics mode,
 *    just cursor addressing capabilities.
 *
 *    It uses os/9 system calls for i/o.
 *
 */

#include <stdio.h>

#define TRUE  1
#define FALSE 0
#define MAXX    80        /* horizontal size of board */
#define MAXY    23        /* vertical size */
#define MAXTARG   20      /* no. of targets per game */
#define ISPEED   400      /* initial ball speed */
#define SPEDINC  100      /* incr/decr in ball spoed */

/*
 *    location of status strings
 */

#define BESTX    70        /* best score so far */
#define BESTY    0
#define MSPS    960        /* "millisecs" per second */
#define SPEEDX    10        /* current speed setting */
#define SPEEDY    0
#define TARGX    30         /* targets */
#define TARGY    0
#define TIMX     50         /* time */
#define TIMY     0

#define BEL     0x07
#define DELETE  0x7f        /* DELETE restarts game */
#define ESC     0x1b
#define NUL     0x00
#define QUITCH  0x03        /* ^C exits program */
#define SUB     0x1a
#define XOFF    0x13        /* ^S */
#define XON     0x11        /* ^Q */
```

```
    #define VBAR    '|'
    #define HBAR    '-'
    #define SLASH   '/'
    #define BSLASH  '\\'
    #define BALL    'O'
    #define TARGET  '*'

    char bellflag;                  /* turns off/on bell
output */
    char board[MAXX][MAXY];         /* board with current
layout */
    int ballx;                      /* ball positioning
data */
    int ballxv;
    int ballyv;
    int ballyv;
    int best;                       /* best score so far */
    int inchar;                     /* character input
buffer */
    int msecs, secs;                /* timers */
    int newtime;                    /* time remaining */
    int speed, diat;                /* speed control */
    int targleft;                   /* no. of targets
remaining */

    main()
    {
        int i;

        clrscrn();
        puts("\nWelcome to Deflection\n");
        for (i = 0; i < 30000; ++i);
        best = 0x7fff;
        speed = ISPEED;
        bellflag = TRUE;
        while (playgame());
        clrscrn();
    }

    playgame()
    {
        char buff[100];
        int i;
        int j;

        inchar = -1;            /* initially no character
input */
        clrscrn();
        getchx();
        puts("\nType any key to start game (h for help):
");
        i = getchar();
        if (i == QUITCH)
            return(0);
        if (tolower(i) == 'h')
            help();
        inchar = -1;            /* reset input buffer */
        initboard();
        targleft = MAXTARG;
        clrscrn();
        for (j = MAXY - 1; j >= 0; --j)
        {
            for (i = 0; i < MAXX; ++i)
                putchx(board[i][j]);
            putchx('\n');
        }
        outs(SPEEDX - 7, SPEEDY, "Speed: ");
        outs(TARGX - 14, TARGY, "Targets left: ");
        outs(TIMX - 11, TIMY, "Time Used: ");
        if (best < 0x7fff)
        {
            sprintf(buff, "Best Time: %03d", best);
            outs(BESTX - 11, BESTY, buff);
        }
        putspeed();
        puttarg();
        msecs = secs = 0;
        puttime();
        ouch(ballx, bally + 1, BALL);
        while (moveball());
        if (targleft == 0 && secs < best)
            best = secs;
        return(1);
    }

    /*
     *    initialize screen for new game
     */

    initboard()
    {
        int i;
        int j;

        ballx = rand() % (MAXX - 2) + 1;
        bally = rand() % (MAXY - 2) + 1;
        ballxv = ballyv = 0;
        i = (rand() & 2) - 1;           /* i = -1 or +1 */
        if (rand() & 1)
            ballxv = i;
        else
            ballyv = i;
        for (i = 0; i < MAXX; ++i)
            for (j = 0; j < MAXY; ++j)
                board[i][j] = ' ';
        for (i = 0; i < MAXX; ++i)
            board[i][0] = board[i][MAXY-1] = HBAR;
        for (i = 0; i < MAXY; ++i)
            board[0][i] = board[MAXX-1][i] = VBAR;
        board[0][0] = board[0][MAXY-1] = '+';
        board[MAXX-1][0] = board[MAXX-1][MAXY-1] = '+';
        board[rand()%(MAXX-2)+1][rand()%(MAXY-2)+1] =
TARGET;
    }

    /*
     *    move the ball and record deflections
     */

    moveball()
    {
        int i;
        int nx;
        int ny;

        dist = 0;
        i = inchar;
        if (i > 0)
        {
            inchar = -1;
            switch (i)
            {
            case SLASH:
            case BSLASH:
                if (board[ballx][bally] == ' ')
                    board[ballx][bally] = i;
                else
                    putchx(BEL);
                break;
            case 'd':
            case 'D':           /* delete character */
                i = board[ballx][bally];
                if (i == SLASH || i == BSLASH)
                    board[ballx][bally] = ' ';
                else
                    putchx(BEL);
                break;
            case 'f':
            case 'F':           /* faster */
                if (speed < 1000)
                {
                    speed += SPEDINC;
                    putspeed();
                }
                break;
            case 's':
            case 'S':           /* slower */
                if (speed > (SPEDINC + 50))
                {
                    speed -= SPEDINC;
                    putspeed();
```

```
            }
            break;
        case DELETE:        /* new game */
            clracrn();
            return(0);
            break;
        case 'h':
        case 'H':       /* help */
            help();
            return(0);
            break;
        default:
            putchx(BEL);
            break;
        }
    }
    switch (board[ballx][bally])
    {
    case VBAR:
        ballxv = -ballxv;
        break;
    case HBAR:
        ballyv = -ballyv;
        break;
    case BSLASH:
        i = ballxv;
        ballxv = -ballyv;
        ballyv = -i;
        break;
    case SLASH:
        i = ballxv;
        ballxv = ballyv;
        ballyv = i;
        break;
    case TARGET:
        if (--targleft <= 0)
        {
            clracrn();
            return(0);
        }
        puttarg();
        board[ballx][bally] = ' ';
        do
        {
            nx = rand() % (MAXX - 2) + 1;
            ny = rand() % (MAXY - 2) + 1;
        }
        while (board[nx][ny] != ' ');
        board[nx][ny] = TARGET;
        ouch(nx, ny + 1, TARGET);
        break;
    }
    nx = ballx + ballxv;
    ny = bally + ballyv;
    ouch(nx, ny + 1, BALL);
    ouch(ballx, bally + 1, board[ballx][bally]);
    setcux(0, 0);
    if (newtime)
        puttime();
    ballx = nx;
    bally = ny;
    /* now delay for a while */
    while (diet < (ballyv ? 22000 : 10000))
        putchx(NUL);
    return(1);
}

/*
 *    explain the game
 */

help()
{
    clracrn();
    puts("The object of the game is to deflect the
ball\n");
    puts("by using the '/' and '\\' characters so that
it\n");
    puts("hits the target.  The time it takes to
eliminate\n");
    puts("20 targets is your score.\n\n");
    puts("Other characters recognized:\n\n");
    puts("\tf - increases the speed of play\n");
    puts("\ts - decreases the speed of play\n");
    puts("\td - deletes the character at the current
position\n");
    puts("\th - prints this message\n");
    puts("\t^S - suspends execution\n");
    puts("\t^Q - resumes execution\n");
    puts("\tDELETE(rubout) - restarts the game\n");
    puts("\t^C - quit\n");
    puts("\nAny other characters will typically cause
the bell\n");
    puts("to sound.  Typing ^G (bell) will toggle the
flag\n");
    puts("which enables/disables the output of
bells.\n\n");
    puts("Type a character to resume play: ");
    _getchr();
    clracrn();
}

/*
 *    Output character at location x,y
 */

ouch(x, y, c)
int x, y;
int c;
{
    setcux(x, y);
    putchx(c);
}

/*
 *    Position cursor to x,y - then output string
 */

outs(x, y, s)
int x, y;
char *s;
{
    setcux(x, y);
    puts(s);
}

/*
 *    Special variation of putchar
 */

putchx(c)
int c;
{
    if (++msecs >= MSPS)
    {
        msecs = 0;
        ++secs;
        newtime = TRUE;
    }
    diet += speed;
    if (inchar == -1)
    {
        inchar = getchx();
        switch (inchar)
        {
        case BEL:
            bellflag = !bellflag;
            break;
        case XOFF:
            while (inchar != XON)
                inchar = getchx();
            inchar = -1;
            break;
        }
    }
    if (c == '\0')
        return;
    if (c != BEL || bellflag)
        _putchr(c);
    if (c == '\n')
        _putchr('\r');
}
```

```
/*
 *    special variation of getchar
 */

getchx()
{
    int c;

    if (c = _status())
    {
        if (c == QUITCH)
        {
            clrscrn();
            exit();
        }
        return(c);
    }
    return(-1);
}

/*
 *    print speed
 */

putspeed()
{
    char buff[10];

    sprintf(buff, "%03d", speed / 10);
    buff[3] = '\0';
    outs(SPEEDX, SPEEDY, buff);
}


/*
 *    update target count
 */

puttarg()
{
    char buff[10];

    sprintf(buff, "%02d", targleft);
    buff[2] = '\0';
    outs(TARGX, TARGY, buff);
}

/*
 *    update time display
 */

puttime()
{
    char buff[10];

    sprintf(buff, "%03d", secs);
    buff[3] = '\0';
    outs(TIMX, TIMY, buff);
    newtime = FALSE;
}

/*
 *    Special version of set cursor which utilizes
putchx.
 *    This version is for Televideo terminals.
 *    Well, so much for portability.
 */

setcux(x, y)
int x, y;
{
    putchx(ESC);
    putchx('=');
    putchx(23 - y + ' ');
    putchx(x + ' ');
}

clrscrn()
{
    putchx(SUB);
}
```

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01453

TIPS FOR BEGINNERS

Data Types

Of all of the high level languages that I know of, FORTH is unusual in that it has almost no data typing. Normally, unextended FORTH has only the byte and integer data types; even BASIC usually has strings and floats (but not always integers). Other, more "modern" languages, such as Pascal and C have many data types; I won't bore you with a list.

The point of this is that FORTH requires the programmer to pay close attention to just what he is commanding the computer to do, because he will probably not get an error message; instead, the program will crash, or, worse, produce a wrong result. When I first started with FORTH, this confusion of data types gave me more programming problems than any other facet of FORTH; fortunately, I have now gained enough experience that I virtually never have this problem. However, I want to emphasize this potential problem in FORTH which many other languages have prevented.

Now that the warning is given, I want to point out some of the advantages of not having to deal with data types. Yes, indeed, there are advantages to this situation, even though it does require some experience to appreciate.

Text Strings

Data in FORTH is just a string of bytes which can be treated in any way appropriate to the program. As a result, any commands which will work with one data type will work with any other data type. If you doubt this, try the following keyboard command:

```
|-|0  20  TYPE<ENTER>
```

After you have recovered from whatever fiasco that command might have caused to your display, you will have a better appreciation for my point!
It should be obvious that any FORTH command which expects to see text data will treat the indicated bytes as text, no matter what the original intention had been for the subject data. Therefore, it is the programmer's responsibility to insure that the desired printable text is actually where it is supposed to be (note that this problem is not limited to FORTH).
On the other hand, it can be useful to manipulate string data ar-

```
PAD1    PAD  80  + ;
: AA    4  0  DO  KEY  PAD1  I  +  C!  LOOP ;
: BB    PAD1  4  TYPE ;
: CC    4  0  DO  PAD1  I  +  C@  ASCII 0  -  PAD1  I  +  C!  LOOP ;
: DD    4  0  DO  PAD1  C@  .  LOOP ;
AA BB CC CR DD
0123
```

Figure 1. Example of math operations on ASCII string data.

ithmetically. As a simple example, you can convert a text string of decimal digits in ASCII into a string of decimal digits in machine code by simply subtracting the ASCII value of 0 from each one in a looping structure. Of course, you can do this in most other languages, but it is usually inconvenient to get to the data once you have made the conversion. Figure 1 is a simplistic example of what I have in mind.

Using any version of FORTH that I know of, entering the lines from Figure 1 from the keyboard will produce:

```
|-|0123
|-|0 1 2 3
```

You can check out the operation even further by dumping the four bytes of data stored at PAD1 .
You may wonder what possible use there might be for this kind of operation. The one which occurs to me first, because I have used it, is that it is an easy way to get keyboard data converted into a form useful for handling peripherals. There are a lot of other useful treatments of ASCII strings with math operations, and I expect to get around to illustrating them, eventually. If you are in a hurry, let me know and I will push them higher on the priority list for future columns.

Bytes

This is actually where most of the data confusion comes from. Byte data can be particularly confusing for people who are used to working with Intel, Zilog, DEC, etc. because of the order in which 16-bit data are stored in RAM (remember the infamous Heath split-octal?). Fortunately, Motorola systems, which we are concerned with, store 16-bit data in the order which appears logical to me; that is, the most significant byte of 16-bit data is stored in the lower address byte of the pair. Both the correct and incorrect forms for displaying separately the two bytes of 16-bit data are demonstrated in Figure 2. Remember that ? is defined as @ . ("fetch dot"). Go ahead and try both forms illustrated in Figure 2 so that you can see the problems with the incorrect form.

Integer Data

The problems of processing integer data are usually small, since FORTH is really geared toward 16-bit elements. The main thing to keep straight is the byte-order if you find it necessary to switch between machine types. Even this is not a problem if you are not addressing hardware, since FORTH will cover all of the other possibilities.

Boolean Data

Ah, this is where the fun can really come in! All of the FORTH-83 books define TRUE as -1 and FALSE as 0, but that is not the whole story. No matter what the situation, 0 is always FALSE , but TRUE is anything else!

It took me a while to realize just how useful this could be, but I now use it regularly. Unfortunately, this can also result in very confusing code if it's use is not adequately documented. I think that this is one of those FORTH programming conventions which make it so hard for some people to understand FORTH, so be careful where you us use it, if anyone else must understand your code.

The usefullness of TRUE being anything but 0 is very noticeable when you are writing code to be used with any version of FORTH. Only FORTH-83 defines TRUE as being -1, all others define TRUE as 1. Nevertheless, they all still recognize a non-0 value as being the logical equivalent of TRUE .

Here is a way to find the first BL (space) in a string. Figure 3 shows a definition which uses a BEGIN ... UNTIL loop to search from the first address, passed to it on the Data Stack, up to the first time a space is found. The count, beginning with 0, is returned on the Data Stack. A pointer to the first space can be generated by simply adding this count to the original address, or the count can be used in other ways.

	correct form	incorrect form
MSB	(address) C@ .	(address) ?
LSB	(address) 1+ C@ .	(address) 1+ ?

Figure 2. The correct and incorrect ways to address 8-bit data.

: This is a very straight-forward definition with no programming tricks or odd manipulations. I have included it simply to show how easy it is to use boolean functions in definitions. This is one of my "stock" definitions, so I have written it to be as general as possible. That way, it is useful with any FORTH dialect; this is the way I think that virtually all definitions should be written. Of course, I grant the necessity for the rare exception, but I think that they should remain just that.
You have to be aware of what you are doing so that you don't make silly mistakes. As an example, for goodness sake, don't use the phrase

```
!-!( address )  C@  TRUE  =
```

for two reasons: (1) the TRUE = part is redundant, and (2) this can only give the proper answer to two out of the 256 possible cases. As an exercise, I'll let you decide what they are. There is a third problem, namely with transportability, as I mentioned above.
I know, someone is going to say, "But that is exactly the situation I was testing for, so why can't I use this phrase?" My answer is that if you know what you are doing, and adequately comment the situation, the go ahead, but don't blame me if you later have problems.
There is a further potential problem with boolean data. The definition of NOT is different among the various versions of FORTH. NOT is defined as the one's- complement of the value in FORTH-83, and possibly others; but NOT is defined as 0= in fig-FORTH and some FORTH-79's. Therefore, the logical result of NOT isn't always transportable, and this can get you into trouble.

Transportability

I keep mentioning the potential problems of transporting programs between different dialects of FORTH because I feel very strongly about it. There are enough problems in the world of FORTH without generating now ones. Therefore, I have a kind of crusade going to eliminate those elements of code from programs which make it difficult, if not impossible, to transfer programs from one version to another.
A particular agrivation to me is the suggested FORTH-83 word NIP . It can be defined as SWAP DROP , and I can see absolutely no need for it. Its name is certainly not very useful in indicating its meaning or application, and it takes longer to execute than the phrase it replaces. Therefore, it strikes me as a cutesy-poo word that never should have seen the light of day
The main problem with NIP is that it gets used without comment, so you can't use a definition containing it, unless you can figure out what it means, or it already exists in your FORTH. There are other words like this, inspired, particularly, by the F83 from Laxan and Perry. Since F83 has so many exceptions and odd definitions in it, its only virtue is that it is free. I mention it because it was written for the accursed IBM, so it has achieved much undeserved popularity. I refuse to answer any questions about where you can get a free copy!

STOCK DEFINITIONS

I mentioned, above, that FIND-FIRST-BL is one of my "stock" definitions. Since I have made reference to this concept in other columns, perhaps this would be a good time to expand on the subject a little more.
In the eight years that I have been writing FORTH programs, I have come to realize that many definitions get used over and over, even though they are not a part of "standard" FORTH. As soon as I noticed this tendency, I started a disk file of these often used definitions. I have about 200 screens, now, of definitions which I have used in more than one program. I admit that most of them have never been used more than twice, but a few get used in very nearly every one of the longer programs that I write. Often, even if I don't have an already-written definition which does the job, I can modify an existing definition so that it will work. Usually, this modification only involves one or two

```
FIND-FIRST-BL   ( adr – count )      \ RDL
    0                                   \ initialize conter
    BEGIN
      2DUP                              \ working copies
      ( adr count )  +  C@              \ fetch current character
      SWAP  1+  SWAP                    \ increment the counter
      BL  =                             \ is char. a space?
    UNTIL                               \ loop on FALSE flag
    SWAP  DROP                          \ scrap the pointer
    1- ;                                \ decrement the counter
```

Figure 3. Finding the first blank space in a text string.

```
: EXPECT  ( adr input-count -- )        \ RDL032888
    2DUP  BLANKS                         \ clear garbage
    QUERY                                \ string input
    >R  TIB@  SWAP  R>  MOVE             \ move string to adr.
    TIB@  80  BLANKS ;                   \ clear garbage
```

Figure 4. EXPECT for Stearns' COLOR-FORTH.

minor changes, so that I can save a lot of debugging time. The FIND-FIRST-BL definition is a particularly good example of this, since I can use it to find any character simply by changing the BL = to ASCII whatever = .

I will start to publish some of these definitions as I get the chance; however many of them are too hardware-specific to be of general interest. In fact, many are so hardware-specific that I cannot even use them on all of my machines!

I will throw in one more stock definition for users of Stearns' COLOR-FORTH. This is a definition for EXPECT , which is needed to make it easier to convert programs written for other FORTH dialects. Reputedly, you don't need EXPECT with COLOR-FORTH, since you write around it. Well, I hate that rationale Even if it means time wasted during compiling or execution, it is still better to use standard words in definitions, since it means time not wasted in making translations. In any case, the definition is in Figure 4.

I hope that you will start your own library of stock definitions, and that you will feel like sharing some of them with the rest of us.

SOFTWARE USER NOTES

A Tutorial Series

By : Ronald W Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

The 68XXX

Well, here I am again. It looks as though I will have enough material to get a new column going. The only change is going to be a little de-emphasis of the 6809 and a lot more said about the 68008, 68000, and 68020 systems. I'll try hard to be objective and not favor any one system, supplier, software etc. over the others. What I like most about recent developments in the 68XXX area is that there would seem to be a product for every taste and budget now available or soon to be so. I'm going to talk primarily, though not exclusively to the hobbyist end of the spectrum this month. At other times the emphasis will be on the higher end systems.

You might have noticed the ads from Peripheral Technology for their 68000 based computer in kit form (available from Data-Comp, see advertising this issue.) They've cleverly designed it around the shape and peripheral bus of a standard IBM PC or clone mother board. It already is capable of using some of the standard PC accessory boards, in particular the Clone hard disk controller that is available for $69. This kit can be purchased with an IBM clone keyboard and monochrome graphics board so that all you need in addition is a monitor. Alternately you can use a standard terminal. *Humbug*™, the monitor that comes with the system, will even select the baud rate to match your terminal. All you have to do is hit CR a couple of times. When you turn the computer on, it literally whistles at you to tell you it is ready to synchronize to your terminal.

Since I had access to standard terminals, I bought the kit configured with a 20 Mbyte hard disk, box, power supply, and the hard disk controller. I also had a couple of 80 track DSDD floppy disk drives available, and without the floppy drive, the total came to right around $900. In addition, I have access to the Peripheral Technology PT 68K-1A system and the Mustang -020 system which is based on the single board computer from Data-Comp. The —020 system runs OS9. The 68000 system that I have presently runs only SK*DOS. OS9 is now being advertised as available for it. The 68008 system runs both, and you can split a hard disk and use a portion of it for each operating system. Getting from one to the other, however, requires a reset and reboot. These systems are available with various clock rates, higher rates being available for higher dollars. Let me give you a rough idea of the processing power relating it to the 6809. If we assign a 1 MHZ 6809 system an arbitrary speed factor of 1, of course, the 2 MHz version has a relative speed rating of 2. Roughly, the 68008 system will be about 5, the 68000 about 8 and the 68020 pretty nearly 15. If you do a lot of number crunching, the 68020 and companion floating point processor 68881 will give you further speed gain.

Naturally the 68XXX systems all support more memory than the 6809. The -1A system allows 720K or 12 times the 64K of the 6809. The 68000 kit computer can handle 2 Megabytes. Half a megabyte is more than sufficient for getting started. The 68020 system comes with 2 MBytes. The increased memory allows the use of RAMdisk and a disk cache technique. OS9 supports Ramdisk very well. SK*DOS supports both ramdisk and disk cache for use with floppy disk drives. The ramdisk is not a great deal faster than the hard disk on the 68008 system, so it is not of much use if you have a hard disk. It is an excellent use of memory if you want to start out with a floppy or two and save a few hundred dollars on the initial investment with the idea in mind of adding a hard disk later.

There is already a great deal of software available for OS9 in the 68K version (any of the three processors). Microware supplies their good "C" compiler as part of the OS9 package. Windrush offers PLuS, a product parallel to PL9 for the 6809 systems. Certified Software offers OmegaSoft Pascal. Stylo Software has Stylograph available. Data Comp has my PAT

editor. OS9 comes with a relocatable assembler and linking loader if you want to program in assembler. OS9 is a slick operating system with which you really can do more than one thing at a time. The basic —020 system allows four terminals, which implies four users. You can be printing out a listing or running a compiler while doing something else, but in single user mode, I've generally found any reasonable sized program to compile so fast that I really had to scramble to get very far doing some other task before the compile is finished. On the other hand it is very nice to be able to print a 50 page listing while editing something else!

PLuS is more a complete software development system than just a compiler. It has a co-resident editor that allows you to fix syntax errors immediately and recompile. When the compiler detects an error, it quits and returns control to the editor with the cursor within a few characters of the detected error. If it is a simple syntax error, you can correct it with a few keystrokes and compile again.

SK*DOS is more affordable for the hobbyist. With the reasonably priced 68000 and 68008 systems, SK*DOS will go a long way toward keeping the cost to an affordable level for those of us who can't justify the cost of OS9 for hobby or personal use.

SK*DOS has an increasing base of software available for it. Thanks to the efforts of Sid Thompson, who has ported a number of public domain "C" programs to SK*DOS, there is a small "C" compiler, a screen editor and a word processor called NRO that is quite usable. Bud Pass has written a 68XXX assembler that is bundled with the 68000 kit but is not public domain. Palm Beach Software (my debating partner of a couple of years ago, Dan Farnsworth) has an assembler available at a very reasonable price. He also has a screen editor for the monitor / graphics board / IBM keyboard approach on the 68000 kit. (More on this below).

I have just received a copy of version 1.0 of Whimsical from John Spray in New Zealand. Whimsical is a Pascal-like language compiler that runs under SK*DOS. I decided to translate JUST as a first test. After getting that running and having not found any errors in the compiler I decided to translate PAT and get it running under SK*DOS. I have that task completed and I expect that one way or another, it will be available soon. I understand that Sid Thompson is about done with a rather full "C" implementation for SK*DOS also. I don't know if that will be available separately or if it will eventually be bundled with the 68000 kit or SK*DOS. I do know that at some time in the future, John Spray's Whimsical will be available as well. It is not yet fully implemented.

If you will allow me to think back a little, I well remember buying (for the company at the time) a Southwest Technical Products 6809 system with 56K of Ram, a dual serial and a dual parallel port, for just about $1500. The dual 8" floppy disk drives in a box with power supply were some $2500 more including the DMA disk controller board. A 68000 system with 1 Mbyte and a 20 Mbyte hard disk complete for under $1000 is a real breakthrough. Oh, yes, the kit comes with SK*DOS included.

Why would anyone want one of these systems as opposed to an IBM clone or a Mac? Obviously not for the array of software that can be run on those systems. If you want to run spreadsheets or color graphics, those other machines are for you, at least for the time being. If you want to learn about computer hardware and software, the 68XXX machines are in my opinion more suitable. I suppose you could argue that one MUST learn about hardware and software to use one of these systems since at least relatively speaking, so little application software is available for them.

I've convinced myself that I can write at least non-trivial application programs and system utilities in assembler for either OS9 or SK*DOS. The OS9 versions are a little harder but there is a good reason. OS9 being a multi tasking operating system has to have a number of safeguards built in. Suppose there are two users on a system. One of them has spent the last two hours typing text and is about to save the text file (It is a poor practice on ANY system to go that long without saving a file, but let's suppose). Now the other user is a klutz like me, writing assembler utilities and making dumb errors so his program is trying to access memory outside of the area that has been alloued to him by OS9. If the system goes down and requires a hardware reset and re-boot when he makes a dumb error, what good is it as a multi-user system? OS9 does NOT let that happen. The user who did the stupid thing gets his bus error and he is dumped. The other users are protected.

The multi-user aspect of OS9 is also the main reason that I/O drivers are so complex. Suppose you and I are editing a text file and we both decide we want to print something. It would be funny the first time, to see your text and my text merged on alternate lines of the printer output, but not much after that, when we realized that we had to ask each other for permission to use the printer for a while. OS9 handles that nicely. OS9 may seem complex on examination, but it is really very straightforward compared to some of the larger operating systems (such as UNIX).

Of course OS9 works perfectly well with a single user, and that single user can run more than one task at a time. He can print out a long program listing while editing another file. I've done that numerous times with both of the 68XXX systems that run OS9, and never have had the computer miss a character input from the keyboard while accessing the disk to feed the printer.

If you are willing to settle for a single user operating system, realizing that you can only do one thing at a time, you can have simplicity and a much greater understanding of the hardware / software interface. Obvi-

ously there is plenty of room for both kinds of operating systems in the area of 68XXX computers, and I will try to maintain balance in discussing both from time to time. Since OS9 is covered rather well in another column in this publication, I will perhaps cover SK*DOS a little more in order to bring more balance to the 68XXX coverage overall.

If you are just thinking about getting into 68XXX systems from a 6809 system, as I did very recently, you can start a little less expensively and learn for a while before proceeding to a full multi-user system.

Incidentally, let me say that I find it harder and harder to go back and use the old 6809 system after long sessions on one of the new systems. If it were not for having to write programs for my occupation, I probably would avoid the 6809 system altogether at this point, and spend my time developing utilities and software for the new systems. As soon as Whimsical has implemented large arrays I will be upgrading my first PAT in that language to have a larger edit buffer. When floating point is implemented, I'll be writing some new scientific function procedures.

I probably ought to mention here why I, as a design engineer and "systems programmer" am still resisting the tide and not switching to IBM compatibles. I have a number of 6809 programs written in PL9 and assembler that have been translated into a number of languages to run on different systems including IBM compatibles. The new Whimsical compiler which, John Spray assures me has not been optimized with regard to code generation, produces about 15% more object code on the average for the 68XXX than the same program for the 6809. I found that a program translated to Lattice "C" on the IBM compatibles produced just about 6 times as much object code as the 6809 version of the same program. The reason for the company to switch from the 6809 is to get more program space. We are bumping 40K to 48K with our programs. Why would we want to switch to a computer and software that would generate 200K to 240K of object code for those programs when we can switch to a system that will give us 45K to 55K for the same program, and provide great excess capacity with much less hardware?

A little time has passed since I wrote the above. The next version of Whimsical has come and I have PAT running under SK*DOS with an edit buffer of 100K. The edit buffer size is fixed at the time the program is compiled and it may be made as large as will fit in memory if desired. PAT now can edit its own source file as a unit all in memory at the same time. It would now seem that all the major bugs have been located and removed so that it is working reliably. I found that the new version has implemented the "C" style notation for incrementing or decrementing a variable. (I suspect the original version had that feature also, but I wasn't aware of it). That is, rather than var := var+1; you may use var+=1; Furthermore, when this notation is used, the compiler takes a shortcut. (Most probably it adds the increment to the variable in memory without loading it to a register first as it would do if it were going to evaluate an expression and assign the result to another variable). There must have been 200 to 300 instances of this in PAT. I changed nearly all of them, and the object code was reduced by about 7%, from 20,500 to 19,100. I timed some long operations and the execution time seemed to be reduced by a small amount too. I was hesitant to change some assignments that involved expressions, but a later test showed that those work as well, and I will do another edit to catch those too.

Assembler Editor Music

Palm Beach Software in Florida has produced a couple of nice products for SK*DOS. The first is a little straightforward assembler that Dan Farnsworth calls ASMK. This assembler does a little of the housekeeping automatically. For one thing it flags long branches that could be short ones. For another, it automatically determines when one of the "quick" instructions may be used, and generates the proper code for it. That is, if your code has an ADD #3,D0, the assembler uses the ADDQ form of the instruction. If you are using a MOVE instruction, the assembler will make it a MOVEM instruction if necessary. You don't have to use the ADDQ or the MOVEM instruction at all.

As those of you who know Dan might suspect, of course the Assembler is written in Assembler code. It is therefore quite fast. This assembler doesn't have macro capabilities or conditional assembly. If you want or need a nice fast simple assembler with no frills and no bugs, this is a good one. It operates very much like the old TSC ASMB for the 6809, after which it was obviously patterned (Thanks Dan.)

Another product of Palm Beach Software is called KRACKER. As you might suspect by the name, it is a disassembler. It works well and is convenient to use. It disassembles code from a binary object file. First it allows you to dump disassembled code and an ASCII representation of the code to the screen. Doing this, you can determine areas that are used for variable storage or constants such as text strings. You tell the disassembler the limits of those areas and you can look again and see if you have correctly defined them. When done, you simply tell KRACKER to disassemble the file to another disk file and the job is done. Of course you then can edit the disassembled code and insert meaningful variable names and labels. I strongly caution those who are new to assembler that a disassembler is only a tool. A great deal of persistence and some skill are

required to make sense of disassembled object code, particularly code generated by a compiler.

The best thing about both of these products is their price. Dan can't be getting rich on these. Do yourself a favor if you have any inclination to do assembler programming, and latch on to ASMK. If you feel you have the persistence to try some disassembly, you ought to get both.

Dan is also working on his music software. The hardware will be available from Peripheral Technology for their PT68K kit shortly if it is not already. I've told Dan that I would be interested in trying my hand at increasing the capabilities of the music software to add a 5th voice and attack and decay features. One can't adequately represent "extended chords" such as the 9th and 11th, with four voices. A C9 chord for instance is played with the right hand or treble voices on E, G, B flat and D, while the bass note is C. The C11 chord is G, B flat, D and F with the C bass note.

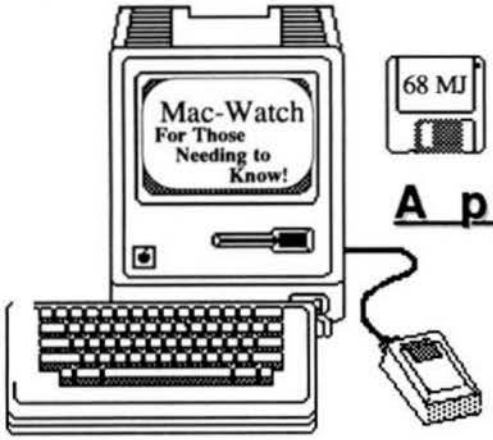
I didn't mention this to Dan, but I'd like to think about a vibrato feature as well. Vibrato is a slight frequency modulation of the notes, occurring at about a 13 Hz rate. I'm not suggesting adding features until the waveform sampling rate is lower than it was on a 2 MHz 6809 system, only some careful consideration to adding a few features at the expense of a little of the sampling rate. Vibrato does a great deal to enhance the sound of electronic music by taking the "computer precision" edge off of the sound. I've simulated it with a rotating organ speaker on my 6809 system, the result being quite organ-like. I look forward to trying the software and looking at the source code. Perhaps it will inspire me to do a whole column on electronic music.

Last note before getting this ready to mail — The company bought another PT68K-2 in kit form and I spent some time on Saturday of the Memorial Day weekend assembling the board. When we fired it up, it didn't work but the problem was quickly traced to one solder bridge on the board. Fixing the bridge made the whole system operational. We plugged in the hard disk interface and booted from the hard disk first try. I spent an hour gathering all the files that I had generated on another system and an evening transferring them to the hard disk of the new system. It is now all configured and ready for action except for an evening's project of installing a DB-25 connector for the parallel port so I can run a printer. Lately I have taken to making all the printer ports on my, and the company's computers IBM printer cable compatible. That way, any printer cable will work on any of our computers. The process is fairly simple, as is making a cable using 25 conductor ribbon cable, a DB-25 male ribbon cable style connector, and an amphenol standard "Centronics compatible" ribbon style connector at the other end.

After a few tries, I seem to have found a combination that is compatible with Epson, Centronics, and a couple of daisy wheel printers. I'll continue this discussion next month for anyone who might be interested.

I might report success with one of my generic cables and the I ran a ribbon cable from the connector on the circuit board to a DB-25 mounted on the back of the chassis. I found the wiring to be 1 to 1 so I could have used a ribbon DB-25 and crimped the connectors on the cable at both ends. However, I had a solder type DB-25 which I connected so that a standard printer cable would work. Except for a single strand of wire shorting two adjacent pins on the connector, I seemingly did it all right. Removing the short made it work immediately.

EOF



The Macintosh™ Section

Reserved as

A place for your thoughts

And Ours.....

Mac-Watch

A Review of Disk Express

By: James E. Law
1806 Rock Bluff Rd
Hixson TN 37343

Getting old is the pits. It takes longer and longer to get up and going. Memory gets fragmented and remembering is not as easy as it used to be. Before you put me out to pasture, let me hasten to say I've been talking about my hard drive. My 20M MacBottom is about a year old and is loaded to the gills. Lately, I noticed that disk-intensive operations have been gradually slowing down. It was time for Disk Express, hard disk optimizing software from AI.Soft, Inc.

The Culprit, Disk Fragmentation

When you load applications and documents to an empty hard drive, the files are arranged contiguously on the disk. This provides for the fastest and most effective hard drive operation since the head travel distance to retrieve the files is minimized. As files are revised, removed, and reloaded, however, it is no longer possible to place the data contiguously. The data is placed wherever the disk directory indicates that there is unused disk space. As a result, a single application or document may be placed in numerous different sections of the disk. Retrieving the files is not a problem since the disk directory accurately maps out the data's location. The problem is just that it takes longer. This phenomena is known as disk fragmentation. How bad the performance of your hard drive is affected depends on the age of the drive, how full it is, and the extent to which files have been loaded and unloaded. For disk intensive programs like PageMaker, disk fragmentation can eventually have a serious affect on overall performance.

Several companies offer disk optimization products but the best known of them is AI.Soft, Inc., who distributes Disk Express at a suggested retail price of $39.95.

What Disk Express Does

Disk Express is designed to reorganize files to eliminate disk fragmentation. Additionally, it performs the following functions:

(a) Detects and repairs some types of directory damage.

(b) Compacts DeskTop—As files are deleted, the finder continues to maintain information about those files. After a while, the DeskTop swells to many times larger than needed. This not only wastes disk spaces, but can significantly slow down the time required for the Finder to present icons and folders when you start up or quit an application. A combination of key strokes can cause the DeskTop to be compacted on any Macintosh but all file comments are lost. Disk Express accomplishes compaction with no loss of information.

(c) Erase Free Space—When files are trashed, any reference to them is removed from the directory, but the files are not really erased. They continue to exist on the hard disk drive until they are written over with new data. Anyone who was determined enough could recover and view these files. In order to secure trashed files from unauthorized eyes, Disk Express can write binary zeroes over all free blocks on the disk. As a matter of fact, for those who are really security conscience, Disk Express can be instructed to write over all free space 3 times!

(d) Disk Express can prioritize disk files, that is, place the files in the order that they are most likely to be used thus reducing access time.

Using Disk Express

To use Disk Express, your Macintosh must be restarted using the Disk Express disk as the startup disk. This is important since the system on this disk is not encumbered with unstable INITS or desk accessories. A system crash during Disk Express could result in extensive data loss so I emphasis again—use only the system provided with Disk Express when you use this program.

When Disk Express is selected and opened, icons are presented allowing you to choose one or more of four functions. These functions are Examine Volume, Compact Desktop, Optimize Volume, and Erase Free Space. The bottom of your screen displays initial information about the disk being optimized including the space used, free space on the disk, and the size of the DeskTop file. A button allows you to review additional information such as the percentage of files that are fragmented and quantity of memory that must be rearranged to optimize the disk. It may take Disk Express several minutes to examine your drive and come up with this additional information.

After clicking the options desired, select EXPRESS from the Special Menu and Disk Express goes to work. You may as well go out to eat, because you are in for a good wait. It took nearly two hours to optimize my 20M hard disk.

I emphasis again, Disk Express must have a stable environment to do its work. Don't let your dog cause a power failure by getting tangled in your computer's power cable or loss of data may result.

Supporting Information

Disk Express works with any Macintosh and almost any disk configuration. It can be used with floppy disks as well as hard disks. The manual adequately describes how the program is to be used.

My Experience with Disk Express

My 20 Meg Mac Bottom was analyzed by Disk Express and after about 15 minutes the following report given:

- 855K free with OK recovered
- 18,318K used for697 files
- DeskTop file occupies 81,042 bytes
- 7% of the files are fragmented
- 17,777K to be moved to optimize

I was surprised that after a year of hard use, only 7% of the files were fragmented. After an hour and 52 minutes, Disk Express completed its work and gave the following report:

- 896 free with 41K recovered
- 18,277K used for 697 files
- DeskTop file occupies 56,060 bytes

The only immediate evidence of Disk Express' efforts was that the finder indicated I had an additional 41K of free space on the hard disk. Luckily, I had recorded the time required to perform a number of tasks both before and after using Disk Express. The following statistics are representative:

Activity	Before	After	Difference
Open Page Maker Doc.	38 sec.	35 sec.	7.9%
Quit Page Maker	12 sec.	8 sec.	33.3%
Open Ready, Set, Go	16 sec.	14 sec.	12.5%
Quit Ready, Set, Go	9 sec.	6 sec.	33.3%

These statistics clearly indicate measurable improvements in the performance of my hard disk. Since my files were only 7% fragmented, the improvement in time required to open files was only modest (i.e., 7-12%). The biggest improvement came in the time required to return to the finder. This 33% speed increase resulted from the compaction of the DeskTop. (Remember, its size was reduced by 41K.)

Recommendations

My experience convinced me that Disk Express is easy to use and does what it says it will do. The performance of my hard drive improved after using this product. I recommend it.

EOF

Logically Speaking

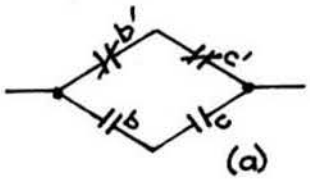
Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

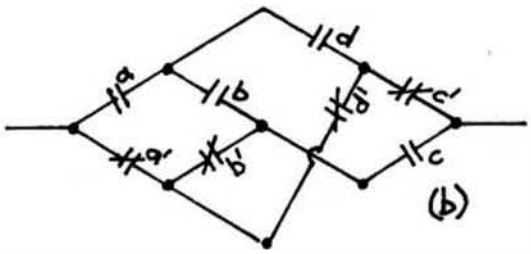
The Mathematical Design of Digital Control Circuits

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

SOLUTIONS TO TEST ELEVEN

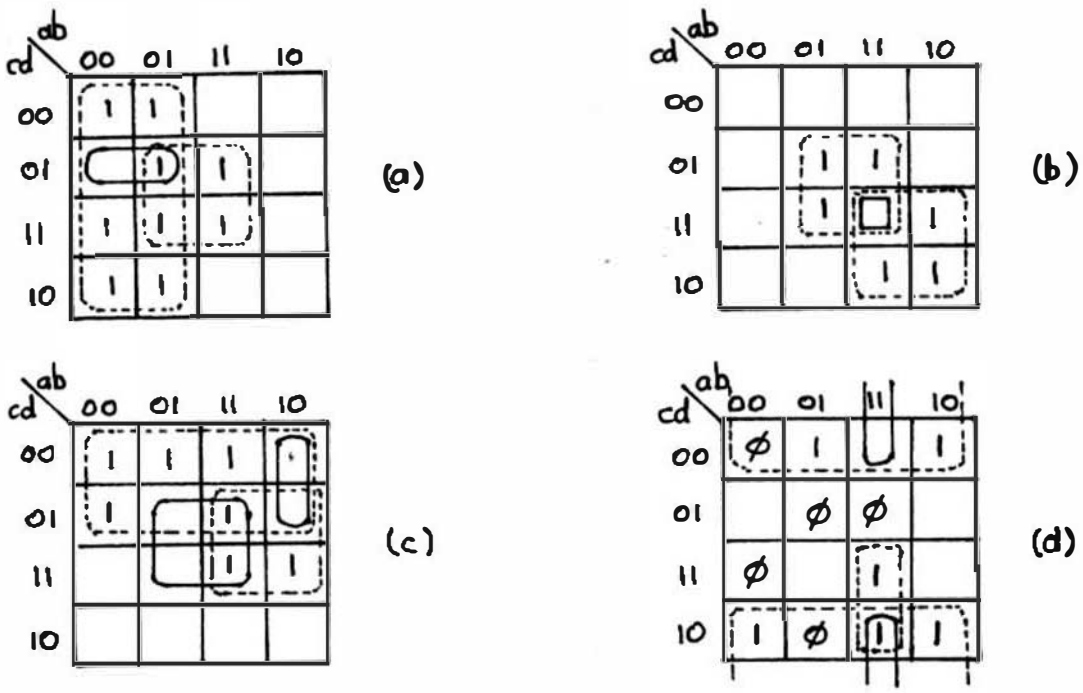
1. Did you notice that the expressions could be simplified? Without this simplification, they'll have proved quite difficult, if not downright exasperating!





2. The factored expressions are given below, together with looped maps to show their derivation

(a) bd + a'(c + d')
(b) ac(b' + d') + bd(a' + c')
(c) c'(a' + b)(b' + d') + ad(b + c)
(d) d'(a' + b') + abc



Mile 14 - heading for Mile 15

Well, we certainly covered a variety of techniques over the last few miles, didn't we? But it was necessary to get them all out of the way so we can look back and feel that, having dealt with the basics of both combinational and sequential circuits, we're now ready for the more advanced design methods that will help to distinguish us as "experts" in this field. First let's take a look at

"BRIDGE" AND "NON-PLANAR" NETWORKS

All our designs so far have resulted in what are known as "series-parallel" networks, that is to say, networks composed of strings of contacts in series or in parallel, or combinations of both. For the next few miles we'll devote our attention more to bridge and non-planar circuits, but before we plunge into all this, I think it would be as well for me to explain in more detail what a "bridge" or "non-planar" network really is. So ... let's look at Diagram 68.

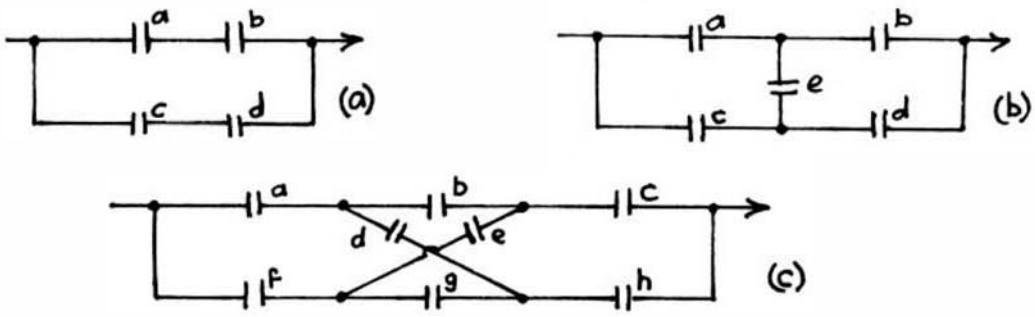


Diagram 68

68a, as we can see, is a simple series-parallel circuit, made up of an a-contact and a b-contact in series, a c-contact and a d-contact in series, and these two strings in parallel with each other.
68b is much more difficult to define. It's the circuit of 68a with an e-contact strung across the middle, and it's

this e-contact which changes the whole picture. For instance, is the e-contact in series with the a-contact in the aed-path, or perhaps in series with the c-contact in the ceb-path? Perhaps neither is true, and the real situation is that ce are in parallel with the a-contact (a + ce), or maybe ae in parallel with the c-contact (c + ae)!
I think you'll agree it's much easier to say the e-contact "bridges" the two parallel paths ab and cd. 68b then is NOT series-parallel, but is a bridge-network. Its true Boolean expression is actually

ab + cd + ade + cbe or a(b + de) + c(d + be)

Finally, what about 68c? Here we have TWO contacts bridging the two parallel paths, but criss-crossed in such a way that if we imagine the input and output lines as stretching to infinity it's impossible to draw this network without a line-crossing. It cannot be drawn on a plane surface without at least one line crossing another, so we call it a "non-planar" network. I'm not even going to TRY to write the Boolean expression for that network!
Having got the preliminaries behind us, we're now all set to venture into the land of

SYMMETRIC FUNCTIONS

In this territory we're going to learn a few new words to add to our vocabulary, and also a strange but surprisingly simple method of designing circuits involving symmetric functions, which will enable us to develop bridge and non-planar networks which will be far more economical than the best series-parallel circuits obtainable by other means. In several cases, we'll produce a finished circuit almost as fast as the specifier can call out his specs. But ... and there always seems to be a BUT somewhere ... before proceeding with the actual design techniques, we have to learn the dialect of this territory and establish the rules of operation of this imponant class of functions.

LANGUAGE OF SYMMETRICAL NOTATION

Let's begin by taking a look at the function

abc' + ab'c + a'bc

That seems to be a perfectly ordinary type of function, doesn't it? Just the sort of thing we're likely to read from a K-map, or to decode from a set of minterms! But there's a lot more to it than meets the eye at this first casual glance. Suppose, for instance, we decided to make the a-variable into a c-variable (and, of course, this means swapping a' and c') and vice-versa. So all "a"s become "c"s, and all "c"s become "a"s, to produce

cba' + cb'a + c'ba

When the variables in each term are arranged in alphabetical order we have

a'bc + ab'c + abc'

and when we compare this with our original expression, what do we find? We haven't changed anything at all, as the two expressions are one and the same. On the other hand if we tried this with, let's say

a'bc + abc we'd end up with abc' + abc

which would NOT be the same thing!
Coming back to our original, we find that this strange relationship holds no matter which two variables we try swapping. The expression is therefore said to be "symmetric in A, B and C", because swapping all possible pairs of the variables does not change the function.
A, B and C are called the "variables of symmetry" rather than just plain "variables".

Another interesting thing about this expression is that the network which it represents will transmit current ONLY IF EXACTLY TWO OUT THE THREE RELAYS A, B AND C are energised. Or, to put it another way, there'll be an output if EXACTLY two of the relays (symbolised by A, B and C) equal 1. This leads us to describe the function as a whole as a "symmetric 2-out-of-3 function of ABC".

In "SYMMETRIC NOTATION" the function is written S^3_2 ABC, and is part of the class of symmetric functions known as "m-out-of-n functions". These m-out-of-n functions may have more than one "m", as in the expression

ab + ac + bc

which, as we can readily see (draw the circuit if necessary), will transmit if any TWO OR THREE of the variables of symmetry equal 1, that is, if any 2 or 3 of the relays A, B and C become energised. In symmetric notation it would be written as $S^3_{2,3}$ ABC.
I don't think you'll experience too much difficulty in understanding that little lot, so let's proceed with

OPERATIONS WITH SYMMETRIC NOTATION

SYMMETRIC FUNCTIONS IN SERIES

$(S^6_{1,2,4,5}\ ABCDEF)(S^6_{2,3,4,6}\ ABCDEF) = S^6_{2,4}\ ABCDEF$

That sure looks a complicated bunch of symbols!! So let's take it a bit at a time, shall we? The left-hand set of parens tells us that we have a symmetric network which will transmit if 1, 2, 4 or 5 of the variables of symmetry equal 1, and the next set of parens encloses a symmetric network which will transmit if any 2, 3, 4 or 6 of its variables of symmetry equal 1. These two symmetric networks are in series - - note the implied AND between the two sets of parens, just as in ordinary Boolean algebra! We are informed that this series network is equivalent to a single symmetric network where an output will be produced if any 2 or 4 of the variables of symmetry equal 1.
Now let's check out the truth of all this! Obviously, considering our two networks which are in series, the whole string can transmit ONLY if both sections are individually transmitting at one and the same time, and as this is true ONLY when 2 or 4 of the variables of symmetry equal 1, then the equation must be TRUE. Another way of looking at this is to say that series networks are AND-networks, and the only subscripts common to both network-1 AND network-2 are 2 and 4. It goes without saying, of course, that the names of the variables of symmetry must also match!!
Go back and read this part a second time, and it'll become clearer.

SYMMETRIC FUNCTIONS IN PARALLEL

$S^6_{1,2,4,5}\ ABCDEF + S^6_{2,3,4}\ ABCDEF = S^6_{1,2,3,4,5}\ ABCDEF$

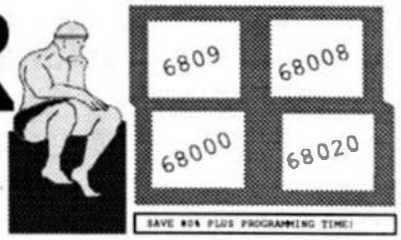
In this example, if EITHER of the two functions in parallel is transmitting then the network as a whole is transmitting. Therefore if 1,2,4,5 OR 2,3,4 of the six variables of symmetry equal 1 the whole network will transmit, which is obviously equivalent to saying 1,2,3,4,5, so the rule is intuitively valid.

COMPLEMENTATION OF SYMMETRIC FUNCTIONS

$(S^5_{2,3,4}\ ABCDE)' = S^5_{0,1,5}\ ABCDE$

Again it should be easy to see that, given five variables of symmetry, and a network which conducts when any 2, 3 or 4 of them equal 1, the circuit will NOT be transmitting when 0, 1 or 5 of the relays are energised, as these

SCULPTOR



From the world's oldest & largest OS-9 software house!

CUTS PROGRAMMING TIME UP TO 80%

6809/68000-68030 Save 70%

SCULPTOR-a 4GL - Only from S.E. Media at these prices. OS-9 levels one and two (three GIMIX) 6809, all 68XXX OS-9 standard systems. Regular SCULPTOR versions 1.4:6. One of if not the most efficient and easy to develope DBMS type systems running under OS-9! A system of flexible keyed file access that allows extremely fast record and data retrieval, insertion and deletion or other programmed modifications. Access by key or in ascending order, very fast. The system provides automatic menu generation, compilation and report generation. Practically unlimited custom input format and report formatting. A rich set of maintenance and repair utilities. An extremely efficient development environment that cuts most programming approximately 80% in development and debugging! Portable, at source level, to MS-DOS, UNIX and many other languages and systems.

Standard Version: 1.6 6809 - $1295.00
68000 $1295.00
68020 $1990.00

Due to a "Special One Time" Purchase, We Are Making This Savings Offer. Quantities Limited!

Once this supply is gone - the price goes back up!

System OS-9: 6809/68000-68030

• Regular ~~$1295.00~~

+ $7.50 S&H USA
Overseas - Shipped Air Mail Collect

ONLY **$295.00**

S.E. MEDIA
POB 849
5900 Cassandra Smith Road
HIXSON, TN 37343 615 842-4601

$AVE - WHILE SUPPLIES LAST!

ECHO echos to either screen or file.

FIND an improved *find* command with "*pattern*" matching and wildcards. Very useful.

HEX dumps files in both hex and ASCII.

INCLUDE a file copy program that will accept "includes" of other disk files.

KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.

LISTDIR a directory listing program. Not super, but better than CAT.

MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.

MULTICOL width of page, number of columns may be specified. A MUST!

PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.

REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!

SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.

SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.

TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.

TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.

UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.

WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source.

Complete set SPECIAL INTRO PRICE:

5-1/4" w/source FLEX - SK*DOS - $129.95

w/o source - $79.95

8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants - - TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/ Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.

Level I OS-9 obj. $79.95, w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk

O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full set of comments and errors messages.

OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS.* It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. **Using HIER a regular (any) FLEX, SK*DOS disk (8 - 5 - hard disk) can have sub directories.** By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modern day solution that all current large systems use. **Each directory looks to FLEX, SK*DOS like a regular file, except they have the extension '.DIR'.** A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!

*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.

*Completely documented Assembly Language Source files included. ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

are the only other combinations permissible. Therefore the complement of 2,3,4 is 0,1,5.
Now brace yourself for the only really tricky part. Make sure you REALLY understand the difference between the next section and the "complementation" just discussed before you continue your journey, OTHERWISE YOU'RE GOING TO FIND IT HARD GOING. I'll try to explain it as carefully as I can!

COMPLEMENTATION OF THE "VARIABLES OF SYMMETRY"

S^{5}_{1},3 ABCDE = S^{5}_{2},4 A'B'C'D'E'

This is NOT the same as the previous complementation! There we complemented the whole function, whereas here we're complementing only the variables of symmetry. Just as in our earlier experiences, we learned to distinguish between complementing an entire function, such as "a' + bc", and complementing a variable, such as "a" or "b".
Let's go VERY slowly here, and begin by looking at the subscript "1" in the left-hand expression. It tells us that the network will transmit current if EXACTLY one of its variables of symmetry equals 1. But isn't this the same thing as saying that it'll transmit if EXACTLY four of its variables of symmetry do NOT equal 1. After all, if we only have 5 relays, and if exactly one of them is energised, it's equally true to say that exactly 4 of them are NOT energised. This accounts for the subscript "4" in the right-hand expression. Similarly, doesn't the subscript "3" (3 relays energised) mean exactly the same thing as 2 relays NOT energised? Hence the subscript "2" on the right of the '=' sign. And, of course, the variables of symmetry are shown complemented to tell us that we're thinking in terms of DE-energised relays here!
The set A'B'C'D'E' is referred to as the "COMPLEMENTED VARIABLES OF SYMMETRY".
And that completes the groundwork for symmetric functions. If you think you've got it all more or less under control, we're ready to study the four different functions set out below.

S^{5}_{1},3,4,5 ABCDE (S^{5}_{0},2 ABCDE)'
S^{5}_{0},1,2,4 A'B'C'D'E' (S^{5}_{3},5 A'B'C'D'E')'

Believe it or not, but all four are equivalent! Let's study the horizontal pairs first, in order to check this out. The top pair says that if the combination 1,3,4,5 relays energised is transmitting then the combination 0,2 relays energised will NOT transmit, ie, 0,2 is a hindering function. The complement of a hindering function is a transmission function, so the complement of 0,2 is the same as 1,3,4,5. Does this seem OK to you?
The lower pair says that if 0,1,2,4 of the relays are DE-energised that this is equivalent to the complement (or opposite condition) of 3,5 being DE-energised. Note that the variables of symmetry are all complemented, meaning that we're considering the condition of being DE-energised, as opposed to the top pair where we're considering the state of ENERGISATION.
Now let's consider the subscripts of the left-hand vertical pair. 1 relay energised is the same as 4 DE-energised, 3 energised is the same as 2 DE-energised, 4 energised the same as 1 DE-energised and 5 energised the same as 0 DE-energised. So they are definitely equivalent! Similarly with the functions inside the parens in the right-hand vertical pair. 0 relays energised is the same as 5 DE-energised, and 2 energised the same as 3 DE-energised. Again, this pair is equivalent. Hence they're ALL equivalent!
Got it? I realise it all looks like mirror-magic or sleight-of-hand, but, believe me, I'm not kidding. As long as you THINK you understand it we can move on. Once we begin using this knowledge it'll become clearer, I'm sure!

THE DESIGN OF SYMMETRIC RELAY-CONTACT NETWORKS

Golly! News sure travels fast in this part of the world! We've hardly begun to study this new field and already we have a client who wants us to design for him a circuit which will activate an alarm-light whenever EXACTLY three out of seven relays are energised. Normally this would be a pretty tough assignment, as it would involve trying to factorise an expression consisting of a total of 35 terms of the form below, where each term consists of 3 uncomplemented variables and 4 complemented.

abcd'e'f'g' + abc'de'f'g' + abc'd'ef'g' ..... + a'b'c'd'efg

What a hassle that'll be, and what a monstrous network!! A 3-out-of-7 network is a symmetric circuit, however, and as we already have our pad of squared paper handy (I hope you haven't lost it along the way!) it shouldn't take us more than about 30-seconds to design the whole circuit. Though it'll take me longer than that to describe to you the design technique involved. The circuit in symmetric notation is

S_3^7 ABCDEFG

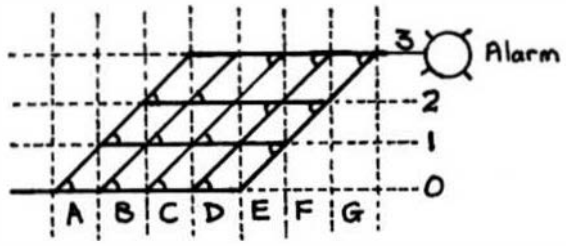


Diagram 69

First of all, we'll quickly label seven columns along the lower edge of an upcoming graph with the designations A, B, C, D, E, F and G in the SQUARES themselves. Then, to the right of the G, we'll label the horizontal LINES with 0, 1, 2 and 3 going vertically from the base-line at G, and with an arrow leading off at the 3-level to indicate the desired output. We'll also add a line leading in at the bottom left-hand corner of square-A to indicate power-input.

Now, commencing at this same bottom-left corner of A at the 0-level, we draw a diagonal line upward to the right until we reach the 3-level, then horizontally to the right to meet our arrow, then diagonally downward to the left until we reach the 0-level, and finally horizontally to the left, back to our starting-point. At this stage we have a hollow parallelogram, and it only remains to fill it with the remaining horizontal and diagonal lines as shown in Diagram 69, AND THERE'S OUR CIRCUIT!

Did you say it doesn't look like much of a circuit to you? That's probably because I haven't yet told you that each diagonal line represents a NO-contact on the relay in whose column it appears, and each horizontal line represents a NC-contact. For the benefit of our client we could, if so desired, draw the circuit along the lines of the small sub-section shown in Diagram 70, where everything is enlarged for clarity.

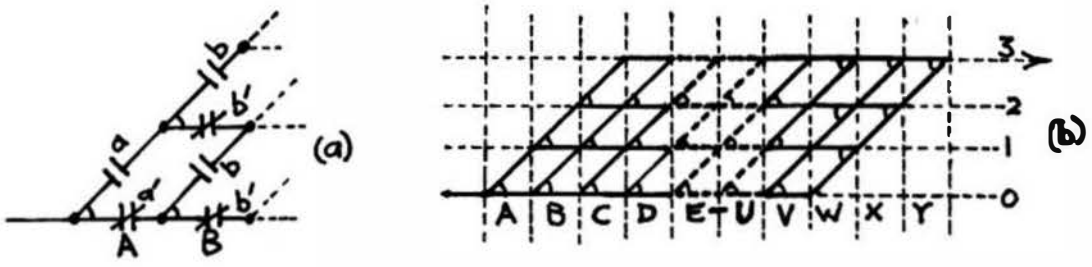


Diagram 70

As it's a symmetric circuit, the DIAGRAM of the circuit in Diagram 69 remains the same, no matter in which order we label the relays along the bottom. Moreover, the contact distribution CANNOT be equalised, the relays at the ends of the diagram always needing less contacts than those towards the centre. Finally, just to make the actual wiring a little easier, it's customary to indicate transfer-contacts with a small arc, as in 69 and 70a.

The circuit-diagram is very easy to interpret. Initially, with no relays operated, power comes in at the bottom-left corner, and goes horizontally along the 0-level via the chain of NC-contacts until it reaches the D-column, and that's as far as it can go.

If any ONE of the first five relays is operated, power will be able to move up to the 1-level, but no further, on its journey to the output at level 3. If, instead, one of the last two relays, F or G, is energised, power will be confined to the 0-level, but the output will now be connected to the 2-level. Either way, input and output will be brought one level closer together!

Even though the following is not strictly TRUE, it's easier to imagine that no matter which relay operates, then power will move up to the 1-level. It really makes no difference, as input and output are still two levels apart instead of the original three, so I'd recommend you use this (slightly incorrect) visualisation.

In like vein, the second relay to operate will direct power to the 2-level, while a third will direct it to the 3-level and horizontally out at the arrow to activate the alarm-light. If a fourth relay happens to operate, power will be directed to a hypothetical 4-level, but as there is no such level, the alarm light will go out!!

If you have any lingering doubts about all this, you should select random combinations of relay-combinations (remembering to switch the contacts in the appropriate columns), and verify that only for EXACTLY three relays will the power input be able to find its way to the output.

Even if the design had called for 3-out-of-50 or more, the design would have been just as easy, because once we've reached the 3-level, each of the intermediate relays is an exact copy of relay-D until we begin to taper off at the extreme right, so we just draw the first four relays, then a dotted section followed by the last four relays, as shown in Diagram 70b for 25 relays.

SYMMETRIC CIRCUITS WITH MULTIPLE SUBSCRIPTS

Our client was evidently well-pleased with our last design, because here he is back again with another project. This time he requires an alarm-light to come ON if EXACTLY 2 or EXACTLY 4 relays out of 5 (A, B, C, D and E) become energised. This we'll translate into symmetric notation as

$S^{5}_{2,4}$ ABCDE

which we read as a "SYMMETRIC 2 OR 4-OUT-OF-5 FUNCTION OF ABCDE".

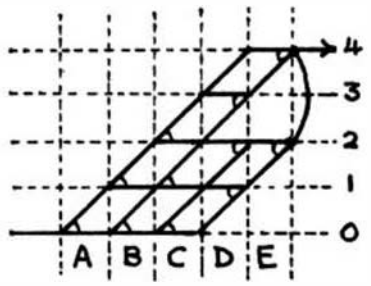


Diagram 71

This turns out to be no more difficult than the last assignment. The procedure is exactly the same, except that we draw TWO parallelograms as if for two separate designs (one to the 2-level and one to the 4-level), but superimposed, with the two outputs joined by a CURVED line. Curved lines represent direct wiring in our new notation. Diagram 71 shows the network for this job, and is particularly easy to follow in relation to the specs.

Any number of outputs may be connected together in this way, but generally there are more economical circuits to achieve this end, as we'll see shortly when we come to the more advanced techniques of symmetric-circuit design.

REDUNDANT TRANSFER CONTACTS

"Redundant" is a word much used in switching theory to mean "surplus to requirements". When we're called on to design circuits whose symmetric notations have multiple subscripts WITH A DIFFERENCE (OR "PROGRESSION") OF 1, redundant transfer-contacts can be eliminated, as shown in Diagram 72.

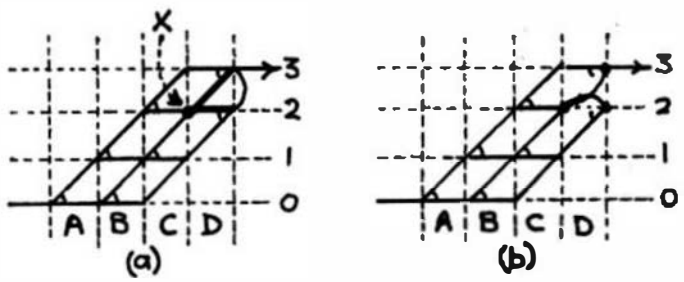


Diagram 72

*72a is the normal initial design for S4/2,3 ABCD. Let's look at point-X at the junction of the two heavy lines at the top-right, which is the focus, as it were, of the curved line joining the two desired output levels. If relay-D is NOT operated, point-X will be connected to the output via relay-D's NC-contact in the 2-level. On the other hand, if relay-D IS operated then point-X will be connected to the output via relay-D's NO-contact leading diagonally to the 3-level. Thus point-X is PERMANENTLY connected to the output no matter what the state of relay-D, so the two D-contacts may be eliminated and replaced with a permanent connection, shown by the curved lines of 72b, thus reducing relay-D to ONE transfer-contact. Note that the two surviving contacts on relay-D do in fact form a transfer-contact, as they have a point in common.
The same principle can be applied to more than two consecutive subscripts, as shown in Diagram 73, which represents the design of the symmetric funtion *S5/2,3,4 ABCDE. Here the points marked Y and Z are first individually made common with the outputs, just as we did with point-X in our earlier example.

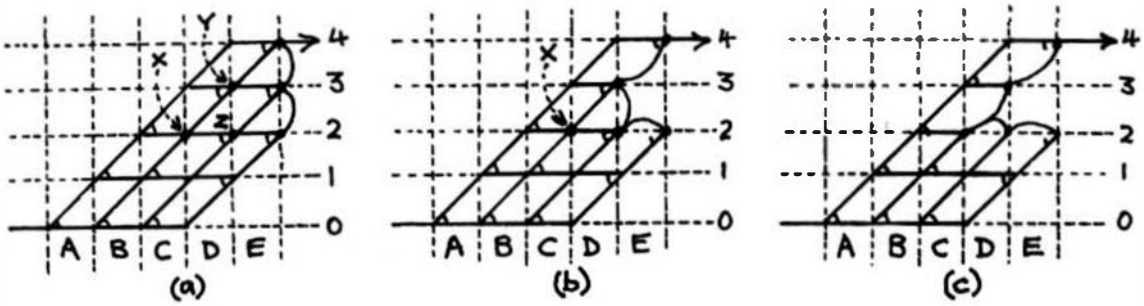


Diagram 73

Now that points Y and Z are permanently wired to the output-line, we find that this in turn focuses on point-X in 73b, which also ends up being permanently connected to the output, as shown in 73c. By this method we've eliminated 1 transfer-contact on relay-D and 2 on relay-E, for a total saving of 3 transfers in this already compact little network! Note in this case, too, that relay-E's NC- and NO-contacts form a transfer, as they have a point in common!

We're now in a position to state all this as a fixed procedure

Whenever multiple subscripts form a sequence with a common difference of 1 (or form a unitary arithmetical progression), a wedge, whose vertical side spans the limits of the common output, may be eliminated from the circuit diagram. The former diagonal and horizontal sides of this wedge are then replaced with permanent connections.

TIME TO STRIKE CAMP FOR THE EVENING

Neat little circuits, aren't they? And SO EASY to design too! Just imagine *trying to design a circuit S25/3,4,5,6 ABCD...WXY by any other method. Anyone crazy enough to try?

Just as an exercise, not an official test at this point, you can try your hand at designing the following. Next time around, when we've studied things from a few different angles, I'll ask you to do these again, using advanced techniques where possible. OK then, try these

(a) S^{5}_{0},2,5 ABCDE (b) S^{8}_{3},4,5,6 ABCDEFGH (c) S^{6}_{1},4 ABCDEF
(d) S^{7}_{1},4 ABCDEFG (e) S^{12}_{3},10 ABCDEFGHIJKL (f) S^{6}_{1},2,4,5 ABCDEF

Who'd have thought just a short while back that you'd even be able to read and understand those little expressions above, let alone whomp out a circuit looking something like a fish-net, all in a matter of seconds? I'd advise you to study example-f carefully in the light of our final procedure above!
Maybe I should also give you a hint regarding the 0-level of exercise-a, in that ALL outputs must initially occur at the right-hand edge of your graph, so the 0-level MUST be taken right through to relay-E, and then linked to the corners of your parallelograms for levels 2 and 5. In effect, the 0-level is a compressed parallelogram, which you've created by following the same rule as for any other level. That is, you've started at the lower-left of relay-A (0-level), moved up diagonally and to the right to 0-level (ie, you've stayed put), then moved horizontally right to the right-hand edge, diagonally down and left back to the 0-level (stayed put again) and finally moved horiazontally left back to your starting-point.
If you'd like to read more about symmetric functions I recommend you read "Switching Circuits for Engineers" by Mitchell P. Marcus, published by Prentice-Hall, 1962. Have fun, and I'll see you next time at this Mile-post!

... End of Mile 14. At marker "Mile 15"

+++

C'ing the States

By J. A. Woodward
556 West Main St.
Lock Haven, PA

This is a program that I had previously written in BASIC. I must say that when one gets the hang of C, it is a lot easter to write such a program because of modularity.

I had great hopes for the Cocolll and level_2 OS9, but as of May 11, I have been unable to get the programmer's package, which contains among other things the level_2 assembler.

This program was written in such a manner that the questions and answers could easily be changed to form a new quiz program. A series of information quiz programs would, I think, be especially marketable to senior citizens, and the Cocolll would be an affordable and powerful enough computer for such a market.

I think that this program is documented enough so as to not warrant a whole lot to be written about it. I have used a structure and have learned a little about pointers, as the function, question(), is used to point variables to the desired information. As I see it, a beauty of pointers is: the easy accessibility of data without having to physically move it from one location to another.

One son suggested that I handicap the answers, but I pointed out to him that the western states' capitals were not so difficult for those who live in the west. A bad thing about this program is that many will not try it because they don't want appear ignorant. I myself have found that it is easier to address envelopes now that I know the correct Post Office abbreviations for the states.

LISTING STATES.C

```
/* ****************************************************************

        Name:       States
        Date:       87/5/11
        Author:     J. A. (Jim) Woodward
        Compiling:  ccl  states.c
        Compiler:   Microware C Compiler
        ********************************

        Function: This program is an educational tool designed to
        help one learn or remember the states-capitals-states'
        abbreviations relationships.

  *********.*********.*********.*********.*********.********* */

#include <stdio.h>  /* External declarations  */

int nums[50], np, nq, cat, right[3], wrong[3];
```

```
char *quest[6]= {"Name the capital city of ",
                 "Give the abbreviation of ",
                 "Name the state whose capital city is ",
                 "Give the abbreviation of the state whose capital is ",
                 "Name the state whose abbreviation is ",
        "Name the capital city of the state whose abbreviation is "},
     *fact, *rtansw;
struct data {
    char   *st;
    char   *city;
    char   *abb;
    };
struct data sca[50]=
{{"Alabama","Montgomery","AL"},{"Alaska","Juneau","AK"},{"Arizona","Phoenix","AZ"},
{"Arkansas","Little
Rock","AR"},{"California","Sacramento","CA"},{"Colorado","Denver","CO"},{"Connecticut","Hartford","CT
"},{"Delaware","Dover","DE"},{"Florida","Tallahassee","FL"},
{"Georgia","Atlanta","GA"},{"Hawaii","Honolulu","HI"},{"Idaho","Boise","ID"},{"Illinois","Springfield
","IL"},{"Kansas","Topeka","KS"},{"Iowa","Des Moines","IA"},
{"Indiana","Indianapolis","IN"},{"Kentucky","Frankfort","KY"},{"Louisiana","Baton Rouge","LA"},
{"Maine","Augusta","ME"},{"Maryland","Annapolis","MD"},{"Massachusetts","Boston","MA"},{"Michigan","L
ansing","MI"},{"Minnesota","St. Paul","MN"},
{"Mississippi","Jackson","MS"},{"Missouri","Jefferson
City","MO"},{"Montana","Helena","MT"},{"Nebraska","Lincoln","NE"},
{"Nevada","Carson City","NV"},{"New Hampshire","Concord","NH"},{"New Jersey","Trenton","NJ"},
{"New York","Albany","NY"},{"North Dakota","Bismarck","ND"},{"New Mexico","Santa Fe","NM"},{"North
Carolina","Raleigh","NC"},
{"Ohio","Columbus","OH"},{"Oklahoma","Oklahoma
City","OK"},{"Oregon","Salem","OR"},{"Pennsylvania","Harrisburg","PA"},{"Rhode
Island","Providence","RI"},{"South Carolina","Columbia","SC"},{"South Dakota","Pierre","SD"},
{"Tennessee","Nashville","TN"},{"Texas","Austin","TX"},{"Utah","Salt Lake
City","UT"},{"Vermont","Montpelier","VT"},
{"Virginia","Richmond","VA"},{"West
Virginia","Charleston","WV"},{"Wyoming","Cheyenne","WY"},{"Washington","Olympia","WA"},{"Wisconsin","
Madison","WI"}
};


main()
{

int  j;

 start();
 for (j=1; j<=np; j++)  {
      game(j);
      crs(20,20);
      printf("Press return to continue");
      getchar();
      }
putchar('\xc');
lcrs(6,0);
for (j=1;j<=np;++j)
  printf("\n    Player #%d   Right=%d   Wrong=%d\n\n",j,right[j-1],wrong[j-1]);
}

start()   /*  This function obtains the needed initial information  */
{
int i;

putchar('\xc');
crs(2,10);
printf("Select the number of players, please. 1, 2 or 3");
for (i=0; i<50; i++)
        nums[i]=i;
lcrs(2,66);
while (np!=1 && np!=2 && np!=3) {
  crs(2,66);
  scanf("%d", &np);
  }
crs(4,6);
printf("Select the number of Questions per player, please. 5, 10, 25 or 50");
```

```
while (nq!=5 && nq!=10 && nq!=25 && nq!=50) {
    crs(4,74);
   scanf("%d", &nq);
   }
crs(6,6);
printf("Select one of the following catagories, please.");
crs(8,6);
printf("     Given               Answer");
crs(9,6);
printf("     **********************************");
crs(10,6);
printf("<1>  State---------------Capital City");
crs(12,6);
printf("<2>  State---------------State's abbreviation");
crs(14,6);
printf("<3>  Capital City--------State");
crs(16,6);
printf("<4>  Capital City--------State's abbreviation");
crs(18,6);
printf("<5>  Abbreviation--------State");
crs(20,6);
printf("<6>  Abbreviation--------Capital City");
crs(22,6);
printf("<7>  Potpourri");
while (cat>7 || cat<1)  {
    crs(22,50);
    scanf("%d",&cat);
     }
cat--;  /* This line is needed because one will select 1 through 7  */
}

randomize( nums, top, amt )  /* This function stores in nums[i], for  */
int   nums[],top,amt;        /* i=0,...,amt-1 integers between 0 and  */
                             /* top-1 (inclusive) 'randomly' and      */
{                            /* without repetition.                   */
    int  i, temp, rtmp;

    for (i=0; amt-->0; i++)
        {
        rtmp = (rand()%(top-i))+i;
        temp = nums[rtmp];
        nums[rtmp] = nums[i];
        nums[i] = temp;
        }
}


rand()      /* This is a pseudo integer random number generator. */

   {
   static  int   n=17;
   char  time[6];
   int  x;

   getime(time);
   x = (time[4]+time[5])*(time[5]+1);
   n=(25173*n+x+13849)%65536;
   n=(n<0) ? -n: n;
   return(n);
}


crs(r,c)             /*  This cursor function is for the CoCoIII   */
     int   r, c;     /*  r=row and  c=column                       */
                     /*  it erases from cursor to end of screen.   */
     {
     putchar('\x1');
     r%=24; c%=80;
     r+=32;c+=32;
     putchar('\x2');putchar(c);putchar(r);putchar('\xb');
```

```
     putchar('\x2');putchar(c);putchar(r);
     }

lcrs(r,c)              /*  This cursor function is for the CoCoIII   */
     int   r, c;       /*  it erases to the end of the line.         */

     {
     putchar('\x1');
     r%=24; c%=80;
     r+=32;c+=32;
     putchar('\x2');putchar(c);putchar(r);putchar('\x4');
     putchar('\x2');putchar(c);putchar(r);
     }


question(qn,temp) /* This function points variables to correct data  */
int qn, temp;
{

   if (temp==0) {
     fact=sca[qn].st;
     rtansw=sca[qn].city;
     }
   if (temp==1) {
     fact=sca[qn].st;
     rtansw=sca[qn].abb;
     }
   if (temp==2)     {
     fact=sca[qn].city;
     rtansw=sca[qn].st;
     }
   if (temp==3) {
     fact=sca[qn].city;
     rtansw=sca[qn].abb;
     }
   if (temp==4) {
     fact=sca[qn].abb;
     rtansw=sca[qn].st;
     }
   if (temp==5) {
     fact=sca[qn].abb;
     rtansw=sca[qn].city;
     }
}

game(gn)
int  gn;
{
char  *add = ", please.", ans[22];
int  i, j, temp, ch;

randomize(nums,50,nq);
temp=cat;  /* This line is necessary for the potpourri category  */
crs(0,30);
printf("PLAYER #%d",gn);
for (i=0; i<nq;++i)  {
  if (cat>5)
     temp=(rand()%6); /*  potpourri category  */
  question(nums[i], temp);
  lcrs(5,3);
  printf("%s%s%s",quest[temp], fact, add);
  ans[0]='\0';
  while (strlen(ans)<=1) { /*  All answers are at least 2 spaces long  */
    lcrs(6,30);
    for (j=0; (ch=getchar())!=EOL;)
       ans[j++]=ch;
    ans[j]='\0';
    clean(ans);
    }
  if  (strcmp(rtansw,ans)==0) {
     ++right[gn-1];
     lcrs(6,0);
```

```
      crs(10,10);
      printf("VERY GOOD");
      }
      else  {
      ++wrong[gn-1];
      crs(6,0);
      lcrs(10,10);
      printf("%s%s.",quest[temp], fact);
      lcrs(11,10);
      printf("The right answer is %s.",rtansw);
      lcrs(12,10);
      printf("Your answer was %s.", ans);
      }
    lcrs(2,20);
    printf("%d RIGHT          %d WRONG",right[gn-1],wrong[gn-1]);
   }
  }

clean(s)  /* This removes blank spaces from both ends of a string.  */
char  s[];
{
int  i, j;

for (i=0;s[i]==' ';i++);
for (j=0;(s[j]=s[i++])!='\0';j++);
for (i=j-1;i>0,s[i]==' ';i--);
s[++i]='\0';
}
```

EOF

8-Bit A/D Converters with SPI

By: David Babin, P.E.
Motorola
MOS Digital-Analog IC Division

The MC145040 and MC145041 are analog-to-digital converters which offer 8 bits of resolution. The total unadjusted error is ±1/2 LSB for these chips. A compact 20-pin package, up to 11 external analog input channels, and a full-duplex serial interface combine to deliver the price/performance designers need. These ADCs feature single-supply operation (5V±10%) and consume only 11 mW.

System on a Chip

As shown in Figure 1, these devices are systems on silicon. The system is comprised of 2 subsystems: the analog subsystem and the digital subsystem as described below.

The chips' analog subsystem consists of a successive-approximation-type converter and an analog multiplexer. The monotonic converter contains an embedded sample-and-hold circuit. Thus, the user's design constraints are eased because slowly-varying signals may be directly fed to the analog inputs. The designer should be aware that the acquisition window is 4 serial data clock (SCLK) cycles plus 6 to 12 µs wide.

The analog subsystem's 12-channel multiplexer has one channel dedicated to an on-chip test voltage. (The test voltage yields a result of 1/2 scale or $80 —hexadecimal 80.) The remaining 11 channels are available at pins 1 through 9, 11, and 12. The worst-case input current requirement is just 400 nA from -40 to 85°C.

The chips' digital subsystem is comprised of an input/output shift register with control logic. The input bit stream is the analog channel to be converted next. The values range from $0X for channel 0 to $AX for channel 10. The test channel is $BX. ("X" is a don't care nibble.)

The output stream is the previous conversion result. The 256 levels range from $00 through $FF and are ratiometric.

The serial port is full duplex. A new channel to be converted is shifted in while the previous channel's conversion result is simultaneously shifted out. Input and output logic levels are compatible with CMOS, NMOS, and TTL.

MC145040 or MC145041

The difference between the two devices is as follows. The MC145040 requires a continuous external clock to drive the successive approximation routine. The advantage is a fast conversion in 10 µs with a 2 MHz clock. The easier-to-use, but slower, MC145041 has an on-chip clock oscillator and an EOC (end-of-conversion) output. The on-chip clock oscillator requires no external components.

A comparison of the two parts is given in Table 1. The maximum sample rates listed take into account both the conversion time and the serial data transfer interval.

Table 1. Comparison of MC145040 and MC145041

DEVICE	CONVERSION TIME	MAXIMUM SAMPLE RATE	COMMENT
MC145040	10µs	41 ks/s	Requires external clock
MC145041	20µs	25 ks/s	On-chip clock

Digital Design Considerations

To facilitate design, all dc and ac digital parameters are guaranteed over a supply range of 4.5 to 5.5 V and an ambient temperature range of -40 to 85°C for plastic-encapsulated devices.

Direct interface to CMOS or NMOS processors is possible. These ADCs are designed to be placed on an SPI bus shared by other peripherals. Eight serial interface clocks are required for the full-duplex transfer.

Use of two 0.1 µF bypass capacitors is recommended. One is placed across the VDD and VSS pins. The second should shunt the Vref and VAG pins. Locate these capacitors as close to the ADC as possible.

Analog Design Considerations

All analog parameters are guaranteed across a supply range of 4.5 to 5.5 V and temperatures from -40° to 85°C for plastic units. Two grades of devices are available. The more-costly devices with a "1" suffix (such as the MC145040P1) accommodate references from 2.5 V to VDD. With a 2.5 V reference, suffix "1" devices offer about 10mV-per-bit steps.

The lower-cost suffix "2" parts use references of 4.5 V to VDD. Suffix "2" chips achieve a step size of approximately 20mV per bit. For suffix "2" devices, Vref may be tied to VDD and VAG tied to VSS. Per Figure 2, these connections are made at the power supply, not at the chip.

Analog input voltage ranges are indicated in Table 2. Note that the devices are functional even if the inputs go 0.8V beyond the VDD and VSS supply rails. With input voltages between the reference voltage (Vref) and analog ground (VAG), the A-to-D conversion result is ratiometric.

To adequately charge the on-chip sample-and-hold circuit, the source impedance driving the ADC's analog inputs must be ≤10kΩ. Unused analog inputs must be tied to VAG (analog ground) to minimize noise pickup.

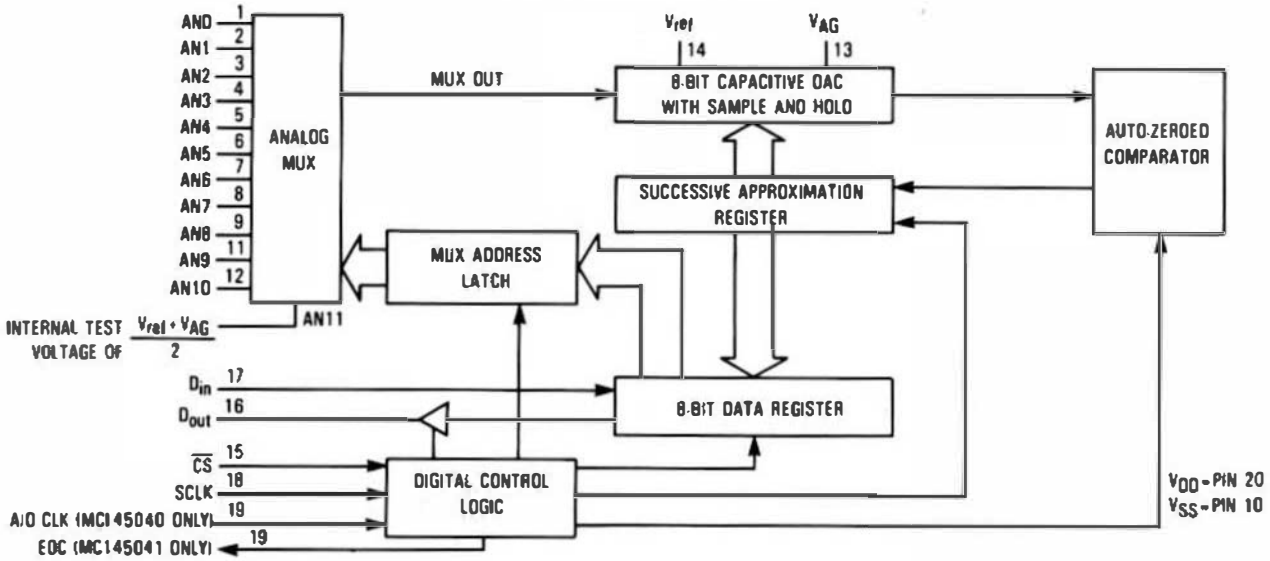
Software Considerations

The software flow for data acquisition is straightforward. As an example, consider that 8 analog channels (AN0 to AN7) are used. AN8, AN9, and AN10 are unused and therefore tied to VAG. The 8 inputs are scanned by reading the digital value of the previously-addressed channel into an MCU and sending the address of the next channel to be read to the ADC, simultaneously. All 8 channels may be scanned in a minimum of 192 µs for the MC145040 or 320 µs for the MC145041.

If the design is realized using the MC145040, 32 A/D Clock cycles (at pin 19) must be counted by the MCU to allow time for the A/D conversion.

The designer utilizing the MC145041 has the end-of-conversion signal. EOC may be used to generate an interrupt, which is serviced by reading the serial data from the ADC and sending the next channel address to be converted.

Figure 1. Block Diagram



Packaging

These 20-pin ADCs are available in plastic DIPs and PLCCs (plastic leaded chip carriers). The plastic packages are guaranteed over -40° to 85°C. For -55° to 125°C operation, CERDIP is offered. NOTE: CERDIPs are subject to a minimum order quantity of 5000 units.

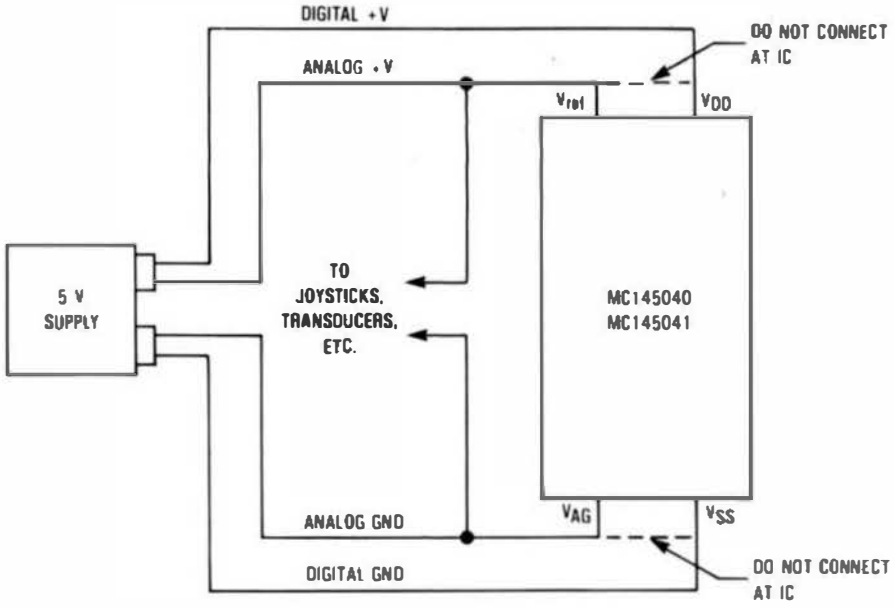
More Info

For a data sheet, call 1-800-521-6274. Ask for #MC145040/D.

Table a. Converter Input Ranges

Analog Input Potential	Conversion Result
	May not function
VDD+0.8V	
	Errors of several LSBs
VDD	
	sFF (overrange region)
Vref	
	Ratiometric to input
VAG	
	s00 (underrange region)
VSS	
	Errors of several LSBs
VSS-0.8V	
	May not function

Figure 2. Using the Digital Supply for the Reference



DO

A FLEX-09 BATCH FILE PROCESSOR

Dave Howland
84 Sedgmoor Road
Flackwell, Heath Bucks
HP10 9AP, England.

1. INTRODUCTION

Flex, while being an excellent operating system, has very limited facilities for command file execution. The EXEC command provided with Flex will only operate on a sequence of other Flex commands in a text file with no way of passing parameters into the file and no control structures to modify the sequence of execution of commands in the file.

DO attempts to overcome some of these shortcomings. It originated from a utility of the same name written by Ben Slaghekke which appeared in 68 Micro Journal in January 1985, but since then it has been extensively re-written to incorporate a number of new facilities. However, my thanks to Ben for the original idea.

In the description that follows, the file of commands operated on by DO will be referred to as a batch file.

2. FEATURES

The main features of DO are :

- up to 9 command line parameters can be passed into the batch file;

- the sequence of execution of the batch file can be modified dependent on the command line parameters, the existence or non-existence of a disk file, or whether an error occurred in the last Flex command;

- Flex command errors can be ignored or can cause the batch file to continue execution from a specified label;

- echoing the contents of the batch file to the terminal during execution can be disabled or enabled as required;

- comments and notes to the user can be incorporated in the batch file;

- batch files can be paused until a terminal key is pressed;

- batch file execution can be aborted from the keyboard;

- batch files may be chained;

3. BATCH FILE INVOCATION

A batch file is invoked by the following command line :

DO <batch file> [<parameter 1> <parameter 2> ... <parameter n>]

If you don't specify a drive number or an extension for the batch file name, DO will use the default work drive and the extension '.BAT'. Parameters can consist of any text string, separated by one or more spaces. To insert spaces into a parameter, the text of the parameter can be enclosed by single or double quotes.

For example, the following are valid parameters :

1.TESTFILE.TXT params.bak 'dcopy.src dcopy.cmd +ls' "Dave's disk" ""

Note that parameters may be entered in upper or lower case, and they will be passed into the batch file exactly as typed. The last example above is used to specify an empty parameter.

Batch file execution may be aborted by entering one or more Control-C characters from the keyboard. These will only be acted upon by DO after any currently executing Flex command has finished.

4. BATCH FILE FORMAT

Batch files are simply text files containing lines of text which may be one of three types :

- Flex commands;

- Flex command parameters;

- DO commands.

Flex command and command parameters are put into the batch file exactly as they would be typed at the console. Thus, if a DO batch file consisted only of Flex commands, it would execute as if it were run by the Flex EXEC command.

DO, however, allows more that this. It supports parameters passed from the command line and special control commands which alter the operation of the batch file. This section describes all of these extra facilitties.

4.1 Command Line Parameters

All parameters in a batch file are indicated by two characters, the first always being '%'. Command line parameters are referred to by the second character being a number 0 to 9 (other special parameters are described below). Thus, '%1' to '%9' refer to the command line parameters, with parameter '%0' referring to the name of the batch file itself. Any parameters not entered on the command line are replaced in the batch file by a null string (i.e... nothing). For example, in a batch file called as follows :

DO DEL DCOPY.BAK DISASM.BAK CAT.BAK

the parameters in the batch file would have the following replacement strings :

%0 = DEL %1 = DCOPY.BAK %2 = DISASM.BAK %3 = CAT.BAK %4 to %9 are null strings

Thus, if the following line appeared in the batch file DEL.BAT :

```
delete %1
```

it would be expanded to :

```
delete DCOPY.BAK
```

4.2 Special Parameters

There are three special parameters which may be used in a batch file. These are '%c', '%l' and '%t', which are used to get a character, a line and many lines from the terminal respectively. When either of the first two is encountered in the batch file, DO will output a question mark as a prompt to the user that terminal input is required. It is up to the user to know what is required, for example by sending a prompt string to the terminal.

Character input may be used as in the following example, which checks that the user wishes to delete a file :

```
delete %1 y%C
```

In this case, DO will run the DELETE utility which outputs two requests for confirmation before deleting the file. The first will be satisfied by the 'y', the second results in the user being prompted for a character which is sent to DELETE. Here the user is prompted by the DELETE utility itself.

Line input allows the user to enter any number of characters at the terminal, up to and including a Carriage Return. If the line has been requested at DO command level, the whole line is read before being acted upon. If the line is requested as input to another program, the input read from the terminal and passed to that program character by character either until a Carriage Return is entered, or until the program exits. For example :

```
delete dcopy.bak %l
```

Here, DO would run the DELETE utility, and on seeing the '%l' would request character from the terminal to pass to DELETE. Because DELETE will exit after one or two characters have been entered, DO will stop reading from the terminal before a Carriage Return has been entered and will revert to reading from the batch file.

Text input (%t or %T) allows any amount of text to be input from the terminal to a program loaded and run by DO. Input reverts to the batch file when the loaded program exits. An example of the use of this is to load the editor from within the batch file but to perform the edits from the terminal. Then when the editor is exited, DO continues with the batch file. The following example shows this :

```
edit %1 %t
```

Note that these special parameters are best used with ECHO OFF, otherwise the terminal display can look somewhat confusing.

4.3 Batch File Commands

DO will interpret a number of commands internally which make the batch file more powerful. These are commands which control the execution of the batch file, and provide information to the user as the progress of the file. All DO internal commands must appear first on the line in a batch file (although there may be leading spaces).

In the following descriptions of the commands, some descriptive characters are used to aid understanding of the operation of those commands. They are:

[W | X] indicates that W or X must appear, but not both;

{Y} indicates that Y is optional;

<Z> indicates that Z is replaced by something else.

The commands may be entered in upper or lower case, DO doesn't mind. However, in the examples DO internal commands are shown in upper case; all other input is in lower case.

ECHO [ON | OFF]

This turns on or off the echoing of lines from the batch file to the terminal. The default is ECHO ON.

EXIT

This causes an unconditional exit from the batch file back to Flex (note that the end of the batch file has an implied exit command which does not need to be explicitly present).

GOTO <label>

This causes execution of the batch file to continue at <label>. If <label> doesn't exist, DO outputs an error message and returns to Flex. Labels are any string of printing characters; the label itself is preceded by the character '@' in the first position on a line. For example :

@loop | other commands | GOTO loop

The jump may be forwards or, as in the above example, backwards in the batch file.

IF {NOT} <condition> GOTO <label>

This command checks one of three conditions and either continues execution from the specified label or with the next instruction in the batch file. The presence of the extra word NOT reverses the sense of the condition. The three conditions tested are :

- for the existence of a file; for example

IF EXIST 1.help.txt GOTO printhelp

If the file 1.help.txt exists, DO will continue execution from the label 'printhelp'.

- whether an error occurred in the last Flex command; eg

delete 1.command.bak yy IF ERROR GOTO notdeleted

In this case the DO will continue from the label 'notdeleted' if the DELETE command returns an error, for example if the file command.bak does not exist.

- whether two strings are the same. This is normally used for testing parameters, for example :

IF %1 = help GOTO printhelp

If parameter 1 from the command line contains the string 'help', then DO will continue from the label 'printhelp'; otherwise execution will continue from the next command in the batch file.

This type of test can also be used with the %c command to alter batch file execution depending on user input, for example :

NOTE Do you wish to continue IF NOT %c = Y GOTO exit

If the user enters anything other than 'Y' or 'y', DO will go to the label 'exit'. Note that this type of operation works best with ECHO OFF, otherwise the terminal output looks somewhat peculiar (try it and see).

A word of warning here. DO performs comparisons on single words delimited by spaces, so for success in these tests any command line parameters to be tested should be single words. Also, if there is any chance that the parameter may be a null string, the parameter indicator must be first in the comparison (i.e. '%1 = help' not 'help = %1') otherwise an error will occur. Null strings can be explicitly tested for, but only by adding an extra character to both sides of the '=' so that DO always has something to compare. For example, use :

IF '%1 = ' GOTO nullparam

If the word NOT is present after the IF, the sense of all the above conditions is reversed.

IF [NOT] <condition> THEN ... [ELSE] ... ENDIF

This structured command uses the same conditions as the previous IF command; however it makes the batch file more structured and relies less on GOTOs. The ELSE part of the command is optional, depending on the requirement. The following are examples of its use :

IF %1 = help THEN list help.txt ELSE | perform commands | ENDIF

IF EXIST %1 THEN delete %1 yy ENDIF

IF statements may be nested to any level (actually there is a nesting level of 255, but if anyone exceeds that then the batch file is too complicated!!).

NOTE <message>

The text following the NOTE command is a progress message to the user. If ECHO is ON, the command does nothing as the line has already been echoed to the terminal. If ECHO is OFF, <message> is output to the terminal. Note that <message> may contain parameters which will be replaced by command line parameters before output to the terminal.

ON ERROR [CONTINUE | GOTO <label>]

This command tells DO what action to take when an error occurs in a Flex command. The ON ERROR command is normally used once at the start of the batch file, but it can be used during the file to change the error operation. The CONTINUE parameter tells DO to ignore the error and to continue with the next batch file line; the error can be picked up by the IF ERROR command. The GOTO <label> parameter causes execution to continue from the specified label. For example :

ON ERROR GOTO reporterror | body of the batch file | GOTO exit @reporterror NOTE An error has occurred ... batch file aborted @exit

To Be Continued Next Month

Bit-Bucket

By: All of us

"Contribute Nothing · Expect Nothing", DMW '86

MOTOROLA INC.

Shernaz Daver
Cunningham Communication, Inc.
(408) 982-0400

Dean Mosley
Microprocessor Products Group
(512) 440-2839

MOTOROLA ANNOUNCES 68000 DEVELOPMENT SOFTWARE ON VAX AND MACINTOSH PLATFORMS

Software Tools Enable Wider 68000 Application Development

AUSTIN, Texas, Aug. 8, 1988 — Motorola today announced a complete integrated software development and debug package for its 68000 microprocessor family that runs on Digital Equipment Corporation's (DEC) VAX series and Apple Computer's Macintosh series. The software, called ToolWare™, brings Motorola's development system software to two of the industry's most popular computing platforms. ToolWare will enable hardware and software engineers to develop more applications for 68000-based systems.

Using the VMS operating system, users of the VAX series can employ ToolWare to develop applications for the 68000/008/010/020 microprocessors. According to Dataquest, Inc., a market research firm based in San Jose, Calif., the VAX dominates 18 percent of the engineering computing market. Using the Macintosh II, SE and Plus with the Apple Operating System, new designs can be developed for 68020-based systems. Development of ToolWare for the 68030 processor is underway at Motorola and will be ready for beta testing at the end of the year.

Interface with Motorola's HDS-300 Development System

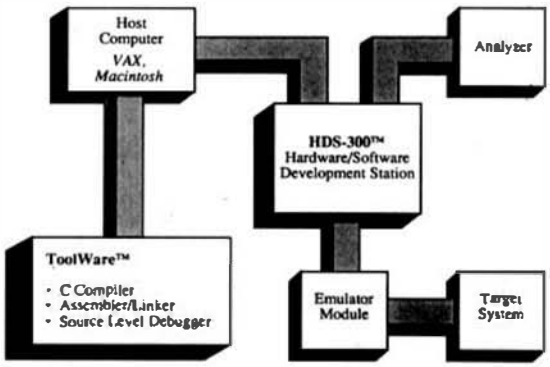
ToolWare allows VAX and Macintosh users to interface with Motorola's HDS-300 development system (see diagram 1), a platform that supports application development for Motorola's 68000 family of microprocessors. ToolWare includes a cross C compiler, assembler/linker and source level debugger with a special feature that allows for quick code compilation, called MakeFile. This feature eliminates the need to specify parameters for each compilation, thus shortening programming time. The source level debugger also allows users to work in C language, instead of assembly language, which makes debugging easier and saves time.

"The 68000 has the largest installed 32-bit hardware and software base," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "With our new Toolware and the HDS-300, VAX and Macintosh users will add to the existing base of software for the 68000 family."

Motorola's HDS-300 development system consists of the following:

- Host computer — The computer that users work on, the VAX or Macintosh.
- ToolWare — C compiler, assembler/linker and source level debugger.
- Development Station — A platform that aids in microprocessor applications development. Motorola's HDS-300 Hardware/Software Development Station supports all the microprocessors in the 68000 family.
- Target System — A system, such as a computer or an add-in board, that is based on a 68000 processor. Hardware and software applications are written for the target system.
- Emulator Module — Hardware that directly plugs into the socket for the 68000 chip in the target system. The emulator duplicates the chip's functions and provides extra circuitry not found on the target system allowing programmers to monitor changes as the software is run. Motorola markets emulators for each microprocessor in the 68000 family.
- Analyzer — Hardware, similar to a logic analyzer, that captures the activities of the chip in the target system for analysis and review by the user on the host computer.

Motorola's HDS-300 Development System



ToolWare for the VAX is priced at $11,200. Individual components are priced at $5,800 for the C compiler, $2,950 for the assembler/linker and $2,490 for the source level debugger. For the Macintosh, the package is priced at $2,800. Individual components are priced at $950 for the C compiler, $600 for the assembler/linker and $1,250 for the source level debugger.

ToolWare is available now from the Motorola Microprocessor Products Group. The development station, emulator module and analyzer for the HDS-300 development system are also available from Motorola.

Motorola's $2.2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America with a balanced portfolio of over 50,000 devices.

MOTOROLA CONSOLIDATES 68000 AND 88000 OPERATIONS

Tom Gunter Responsible for All Development, Design and Marketing

AUSTIN, Texas, Aug. 9, 1988 — Motorola today announced the consolidation of all development, design and marketing activities for its 68000 and 88000 microprocessor families within its High-End Microprocessor Division. The division, a segment of the Microprocessor Products Group, is headed by Tom Gunter, vice president and general manager. Gunter, 41, has been with Motorola 13 years. He is widely credited for the development and success of the 68000 family.

"Tom's experience in creating and managing successful microprocessor operations is recognized throughout the industry," said Murray Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group. "With Tom's leadership, we look forward to even greater success with our entire product portfolio."

The 88000 and 68000 families are both high-performance microprocessor lines that address the general-purpose computing market. The recently introduced 88000 family incorporates many features using RISC (reduced instruction set computer) technology that will allow it to expand markets such as fault tolerance and multiprocessing. The new

family, developed within the Microprocessor Products Group by a dedicated research organization, has been consolidated into the High-End Division as it approaches volume production.

The 68000 family continues to be the dominant force in the workstation and embedded controller markets. It powers over $100 billion worth of computer hardware and has the world's largest 32-bit software base. More than 18 million 68000 family chips are installed worldwide in systems belonging to leading computer manufacturers including Apollo Computer, Apple Computer, Hewlett Packard, IBM, NCR, Sun Microsystems and Unisys. It has gained wide acceptance in embedded control applications such as computer peripherals, factory automation and telecommunications. The High-End Division is currently working on the 68040 processor, the latest in the series of 68000 family compatible chips.

EDITORIAL CONTACT
Shernaz Daver
Cunningham Communication, Inc.
(408) 982-0400

ORDERING CONTACT
Motorola Literature
Distribution Center
(602) 994-6561

MOTOROLA ANNOUNCES 68000 FAMILY REFERENCE GUIDE AND UPDATED USER'S MANUAL

Publications Address Wide Range Of Processor Applications

AUSTIN, Texas, Sept. 6, 1988 — Motorola's Microprocessor Products Group today announced a family reference guide and an updated user's manual for its 68000 microprocessor line. The publications provide customers with information on Motorola's microprocessors and related products used in applications ranging from embedded controllers in robotics to processors in supercomputers.

The 68000 Family Reference Guide gives system designers and end users a technical overview and description of the 68000, 68008, 68010, 68020 and 68030 processors. The guide also includes information on 68000-based coprocessors, direct memory access (DMA) controllers, data communications devices, and hardware and software evaluation systems.

The 68000 User's Manual, targeted at systems designers, programmers and software developers, consolidates detailed information on creating hardware and software for the 68000, 68008 and 68010 microprocessors. Information on the 68020 and 68030 processors is contained in separate user's manuals.

H.C. ANDERSEN
COMPUTER A/S

Englandsvej 380
DK-2770 Kastrup
DENMARK
Tel. (45) 1 52 44 04

68 MICRO JOURNAL PBC
5900 Cassandra Smith Road PO 849
Nixon,TN 37343, U.S.A.
att: Mr. Don Williams Sr.

PRESS RELEASE

ATARI MEGA ENTER SIX MULTIUSERS

H.C.ANDERSEN COMPUTER A/S of KASTRUP, DENMARK can now offer expension cards and OS-9/68000 Serial drivers for the ATARI MEGA 2/4.

The ATARI MEGA now enters computers in the UNIX family with full multitasking and multiuser environments.
The maximum amount of users on a ATARI MEGA 2/4 is six. ATARI 520/1040/MEGA or PC's can be hooked up as userterminals.
The expansion card is based on I.ST CARD BASIS and one or two I.ST CARD SIO, each with two serial ports. I.ST CARD BASIS fit direct to the ATARI MEGA internal expansion port and it is easy to install.

PRICES	US DOLLARS
I.ST CARD BASIS	270,00
I.ST CARD SIO with two RS 232C ports	170,00
OS-9 Operating System Personal V2.2	295,00
OS-9 CARDSIO Driver/descriptors t2-t5	265,00

Prices is excl. VAT

OS-9/68000 is an operating system in the UNIX family. It is supported by BASIC, FORTRAN, C, PASCAL, TEXT PROCESSING, CALCULATION and 4GL SCULPTOR.

H.C.ANDERSEN COMPUTER A/S has delivered the first ATARI MEGA MULTIUSER SYSTEM for DATABASE purposes.

Sincerely,
H.C.ANDERSEN COMPUTER A/S

Hans Christian Andersen
Manager, BC

68000 family processors were the first to offer a full 32-bit internal architecture. There are 18 million 68000 family microprocessors in the world today, all running 32-bit software. Members of the 68000 family are fully compatible, allowing for easy software migration from one processor to another.

The Family Reference Guide, priced at $1.65, and the User's Manual, priced at $4.45, are available from the Motorola Microprocessor Products Group. To obtain the publications, write Motorola Literature Distribution Center, P.O. Box 20912, Phoenix, AZ 85036, and request order numbers FR68K/D for the Family Reference Guide and M68000UM/AD for the User's Manual. Checks or money orders are payable to Motorola.

EDITORIAL CONTACT:
Bob King
512/928-6163

READER CONTACT:
Julie Kleinhopf
512/440-2035

INQUIRY RESPONSE:
Technical Info Center
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA ANNOUNCES THE DYNAMIC NEW EVALUATION PRODUCTS BROCHURE

Austin, Texas, July 26, 1988.....The Motorola MCU Division proudly announces the newly revised Evaluation Products Brochure. Motorola offers the highest quality development systems for dedicated controllers at the lowest prices ever.

Motorola evaluation products can debug, design and evaluate Microprocessors, Digital Signal Processors and/or Microcontrollers at a fraction of the cost of most other evaluation devices offered by competing manufacturers. Motorola's broad range of evaluation systems offers a very cost effective solution to your application needs. The complete line of single-board EVM's is just $500 each.

The products brochure contains specifications and features of ten different evaluation boards and the devices that they support. In addition to EVB's, the brochure also lists features and specifications of Motorola Development Systems and Hardware Development Stations.

The Evaluation Products Brochure is available immediately. To order your free brochure, please contact Motorola's Literature Center at (602) 994-6561 or write to request document #BR292, REV2 at:

Motorola Inc.
Literature Distribution Center
P.O. Box 20912
Phoenix, AZ 85036-0924

FORCE COMPUTERS, INC.
3165 Winchester Blvd.
Campbell, CA 95008-6567

Product News

Prepared By:
Shohat & Davis PR
2959 S. Winchester Blvd, Campbell, CA 95008
Murry Shohat (408) 379-7434

Contacts:
FORCE USA: Wayne Fischer (408) 370-6711

Running Start™ Program Speeds VME Projects with Software Drivers, C Libraries & Documentation

Comprehensive Support Packages Are An Industry First

CAMPBELL, CA., August 14, 1988 — In a move expected to become standard in the VMEbus industry, FORCE COMPUTERS has launched a comprehensive product support program that includes software drivers and development libraries plus extensive documentation. Called *Running Start*, the program provides high performance 32-bit board level products together with complete I/O device drivers, all from a single source. Engineers and programmers receive a single package containing most everything needed to start a development effort.

The current industry practice among many board vendors is to supply hardware only. In most cases, the software drivers are acquired from vendors of operating systems and kernels or are developed by the customer at substantial time and expense. Since buyers must acquire diverse tools from several sources, projects are slowed and confusion often accompanies the practice. In addition, acquisition costs are often very high due to licensing requirements.

"Our intention is to ease the integration of FORCE products into customer environments," said Wayne Fischer, Director of Marketing. "We placed ourselves in the role of VMEbus system developers, then literally developed hardware-specific drivers and documentation that an engineer would otherwise acquire or create in the early stages of a project."

The acquisition of device drivers is usually the most difficult step. Many companies undertake the writing of custom drivers. In high performance applications where real-time operations are needed, the device drivers play a crucial role. Their development is difficult and time consuming at best, and poorly crafted drivers can literally cause performance degradation beyond acceptable limits.

The *Running Start* package can save several months of development time for new designs which integrate several VMEbus boards. Board-specific drivers are hand-crafted and pre-tested by FORCE. Most are written in "C" conforming to the Kernighan & Ritchie (K&R) standard to enable rapid system integration. All software components in the package have undergone rigorous development and beta test.

FORCE will offer *Running Start* packages with all of its intelligent board level products, including CPUs and I/O boards. The first *Running Start* package is available now for the CPU-29 and CPU-32 single-board computers based on the 32-bit 68020 and 68030 microprocessors, respectively.

"As we introduce new packages for our boards, customer support issues will decline," claims Fischer. "Given the rapid growth in board complexity, everyone will benefit as this program eases integration, speeds products to market and reduces development costs."

***Running Start* Includes Many Tools**

A typical *Running Start* package consists of board-specific device drivers in source code, extensive documentation, over 150 documented subroutines that support board functions, diagnostics and demonstration programs. Unique in each package are detailed examples of hardware-specific source code that are applicable only to the FORCE-supplied board. Typical drivers include serial devices, timer support functions, real-time clock control and disc control, etc.

These examples are extremely useful in two ways. First, the system engineer has code available for immediate use to integrate the FORCE board with specific I/O needs. Second, the examples represent "approved" samples of correctly written drivers that have been rigorously tested. By studying the code structure, system engineers get a "running start" on custom drivers that ordinarily take a long time to write and debug.

Other benefits of the supplied tools include the ability to quickly demonstrate hardware operations, even by non-technical personnel. The tools also offer a consistent interface, thus encouraging standards. All in all, *Running Start* eases integration by sharply reducing the number of problems customers have experienced in the VMEbus marketplace.

"FORCE believes that *Running Start* will reduce the volume of support calls made to us by customers who purchase one of our products for the first time," said Fischer. "In the medium term, this logical program will become the normal way to supply board-level products," he predicted.

Price & Availability

Running Start is available immediately for the CPU-29 and CPU-32 single board computers. It is priced at $490.00 (single copy) and is provided on 5-1/4" PC/MS-DOS compatible diskettes. *Running Start* packages are now under development for many other FORCE boards; each will be announced as it becomes available.

About FORCE COMPUTERS

The leading independent designer and manufacturer of VMEbus products, FORCE is now in its sixth year. The company has completed 22 consecutive quarters of profitable operation. Force is headquartered in Campbell, California with subsidiaries in West Germany, France and the United Kingdom. Sales, service and product support are provided on a worldwide basis.

MICRONICS RESEARCH CORP.

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

Dear Don,

A short list of corrections for 'Logically Speaking' - Miles 11 and 12.

July 88, p 42 caption **Diagram 48** missing from diagram. Also last part of para 5 should be amended to read "... involved, we'll set to 0 the top two cards, namely the 32-card and the 16-card, and put them at the bottom of the heap."

Aug 88. p 20, para 1. "... complement Ll's decoding" NOT "Ll'S decoding"
p 22, just before para beginning 'A "non-planar" network' Diagram 53 should be inserted, together with its caption. (My fault entirely!). This one is important, as supporting text is meaningless without diagram.
Last para of this section should read "... normal ~~left-to-right~~ configuration."
p 23 Diagram 55 - delete (a) and (b) from caption.
p 24, para 3 delete parens, to read "... of power instead of AC, as we've assumed till now."
p 26 Insert one more space between the terms in the expression near top of page.

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

R. Jones
President

68 Micro Journal
5900 Cassandra Smith Road
Hixson,Tn. 37343

Dear Don.

Since there are so few 68xxx BBS's in comparison to other systems I'd like to mention one that I've found which carries a variety of programs so that all 68xxx users should find something to their liking.

The sysop is Michael Evenson and the number is 817-468-8398. Michael lists freeware and shareware programs for such operating systems as 6809DOS, FLEX, OS/9 etc.

The system I have is a 12.5mhz 68000 running 6809DOS/68k which I purchased as a kit from Peripheral Technology. The 6809DOS USERS GROUP has given permission to Mike to list their programs on his BBS such as a screen editor, uEMACS, small-c compiler, cut and paste programs,etc.

He has also written his own programs and placed them as either freeware or shareware on the BBS and they are great! Karc is a phasrc compatable unarcing utility for 6809DOS and is offered as shareware for a $25.00 registration fee. EIM260 is a communications package complete with 30 entry auto-dialer and is offered as shareware for a $45.00 registration fee. Listcpm is a freeware filter program presented so that text files from other formats can be handled by 6809DOS after dl'ing. The shareware programs are strictly voluntary registration. No fee is REQUIRED, but registration provides the user with mailed updates and enhancements for a year from date of registration. Mike has also written a full screen disk editor called DPATCH. Source is available for DPATCH which is written in small-c. The very modest voluntary registration fee requested is a small price indeed when considering the value these programs provide the user.

All in all, this BBS contains a lot of useful programs and is run by a sysop who really cares about furthering the usage of 68xxx based systems.

I highly reccomend that readers try the BBS out and if like me, they live a good distance from Texas, call on the weekend when rates are much lower. The price of the phone call will turn out to be a good investment in quality programs and communicating with other 68xxx users interested in expanding the capability of their systems. This BBS is PC-PURSUITABLE. (C D/TXDAL/12.id or C D/TXDAL/3.id).

On another matter, please send a copy of your author's guide as I am interested in submitting articles and as you say - contribute nothing, expect nothing.

Last, enclosed is a check for a one year subscription as I have to drive 30 miles to find a store which carries Micro68.

Sincerely,
Michael Daly
Michael Daly
334 Main Street Apt.B-4
East Greenville, Pa. 18041

Classifieds

As Submitted - No Guarantees

MUSTANG-020 20Mhz with 68881. OS9 Professional Package & C $2900. Call Tom (615)842-4600.

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.

S+System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $3400.

HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.
3-Dual 8" drive enclosure with power supply. New in box. $125 each.
5-Siemens 8" Disk Drive, $100 each.

Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.
QUME QVT-102 terminal, like new, amber screen $250. or best offer.

SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09CPU Card- $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

***

SWTPC 6809 System S709, Dual 8" drives, 8212 terminal, 128 K RAM, Okidata wide carriage, with tractor feed, Make an offer.
(813) 462-0511 or (813) 536-0018. Pete Yore.

***

Special Liquidation

30 Motorola MVME 133, CPU Module, 68020, 12.5 Mhz, 1 Meg DRAM, $675 ea.
30 Motorola MVME 225-1, 1 Meg Memory Module, $380 ea.
30 Motorola MVME 320A, Winchester/Floopy Controller, ST506 compatible, $1295 ea.
30 Motorola MVME 332, Intelligent8 Channel Serial Communications Module, 68010 supports multiple protocols, $1795 ea.
35 Electronic Solutions 7 Slot VME Desktop Enclosures w/325 watt power supply, supports up to 4 mass storage devices, $2000 ea.
2 Configured Systems, 40 Meg Seagate Winchester, 1 Meg Floopy, $3200 ea.
Call or write J.G. (602) 951-3373, RPG, POBox C6000, Suite 162, Scottsdale, Az. 85261 or
Tom Williams, (615) 842-4600.

***

COCO III, 512, 30 Meg HD, 720K FD, MultiPak, (2) serial ports, mouse, OS9-LII, Multi-Vue, Dev. Pkg, 'C', $1200.
(313) 227-2412 after 6 pm EST.

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga*. As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your **OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microproceasor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERM EM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card #_____ Exp. Date_____

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $_____

Name _____

Street _____

City _____ State ____ Zip _____

Country _____

My Computer Is: _____

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

***Foreign Surface: Add $12.00 per Year to USA Price.**

***Foreign Airmail: Add $48.00 per Year to USA Price.**

***Canada & Mexico: Add $9.50 per Year to USA Price.**

***U.S. Currency Cash or Check Drawn on a USA Bank !**

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime. **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Prnnod, **Snoopy, **Football, **Hexpawn.**Lifetime.
Disk- 3	Cbug09, Secl, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Harkness.
Disk- 8	Crtest, Lanpher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Annstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88Koipi.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple(mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9 FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~$199.00~~

• 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

• 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

• 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program in BASIC, Debug it in BASIC and Then Compile it to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

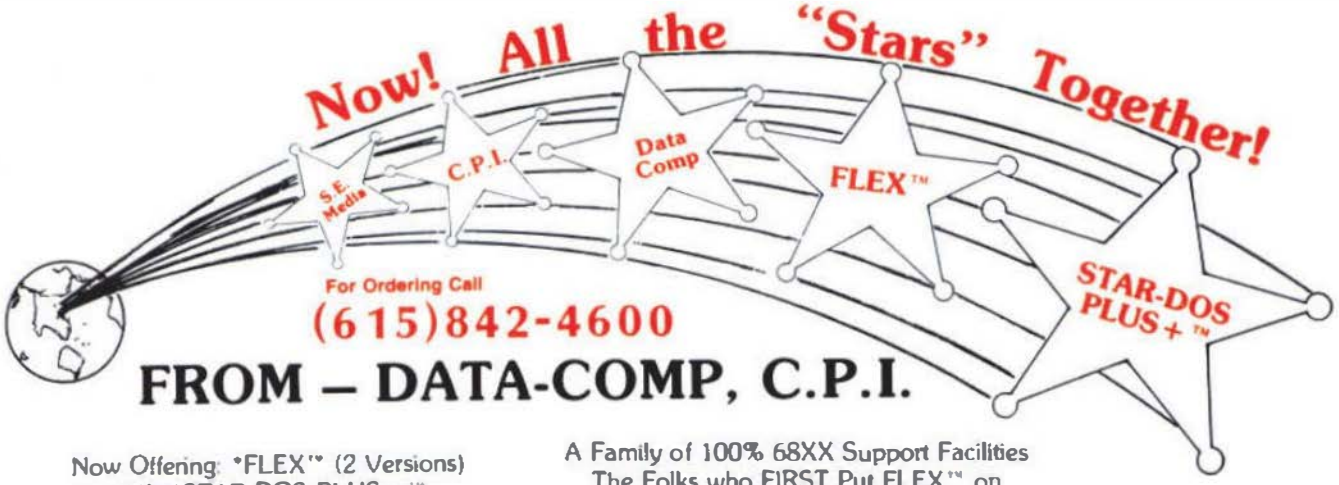
CPI **S.E. Media**™

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.

Price subject to change without notice.



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K
FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O*. Fact is, the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks!

SALE!!

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

Now even faster!
with 12 Mhz CPU

C Compile times: OS-9 68K	Hard Disk
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC — MUSTANG-08 is a trademark of CPI

MUSTANG-08

LOOK

					Seconds	32 bit Integer	Register Long
Other	68008	8	Mhz	OS-9	68K	18.0	9.0
MUSTANG-08		10	Mhz	OS-9	68K	9.8	6.3

```
Main()
{
                    /* int i; */
C Benchmark Loop    register long i;
                    for (i=0; i < 999999; ++i);
}
```

Data-Comp Division

A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC Hi-density FLEX 5" - Call for special info.